

**Release Notes for Thoroughbred/OS™
Version 6.6.2 and Later Versions**

*IBM PC-AT, PC-Xt, PS/2 AND
COMPATIBLE MICROCOMPUTERS*

**Thoroughbred/OS™ Release Notes
Version 6.6.2 and Later**

**COPYRIGHT © 1988 CONCEPT OMEGA CORPORATION
All Rights Reserved.**



This documentation may not be reproduced or modified in any way, without the express written permission of:

**Thoroughbred®
Concept Omega® Corporation**

19 Schoolhouse Road • PO Box 6712 • Somerset, NJ 08875-6712
Phone: 201-560-1377 • Telex: 910-3808-394 • Fax: 201-722-7958

#TRN-070588-662 +

Thoroughbred, Concept Omega, Thoroughbred Business BASIC, Thoroughbred Business BASIC Operating System are trademarks of Concept Omega Corporation.

Upgrade to V6.7 from V6.6.X

Software Enhancements

1. Support is provided for 33 started tasks. Up to 4 multi-port controllers are supported. Also the number of FDTs (open files) which may be configured has been increased to a maximum of 800. *INSTALL/*NPSD can now configure unused memory in the 128-192k bank or configure additional memory in the 192-256k bank as required.
2. Added special variables and the CVT function required by the 4GL language (IDOL-IV).
3. Key files may now be accessed sequentially backward using the PREAD and PEXTRACT directives. The PREAD and PEXTRACT directives follow the syntax of the READ and EXTRACT directives respectively (see the TOS Reference Manual). The LKY, PKY, and FKY functions have been implemented to return the last key, previous key, and first key of the file respectively. These functions follow the syntax of the KEY function in the TOS Reference Manual.
4. Support is provided for up to 3 PS/2 dual-port serial boards (up to 8 users).
5. Support is provided for the Thoroughbred 10Mhz memory board.
6. A task may be released by the control task even if in a public program (ERROR 38 is no longer issued). There is no effect on any ADDRed programs remaining in the bank.
7. The START directive will now give an ERROR 38 if used in a public program to start itself.
8. The REMOVE directive will now give an ERROR 11 if the KEY = option is omitted.
9. The security system will allow up to 7 boots with missing or bad serial devices. This should allow dealers to set up hard disks before going to the end-user site.
10. I-O queue processing has been made more efficient. A time slice feature has been added to reduce overhead in internal task management.
11. The performance of the CRC function has been improved.

12. Time-out has been implemented for printers on serial ports. Any print statement which does not have a time-out will default to a 15 second time-out. This is the same as parallel printers. There is no time-out for attached printers due to memory problems in the terminals. A patch will be made available to change the default time-out values.
13. The new XT-2 bios identification is recognized by the loaders.
14. Thoroughbred now supports DTR and DSR on the IBM Asynchronous Communications Adapter, all ports on the Thoroughbred intelligent expansion board and on the first port on the Thoroughbred standard expansion board.

Software Changes

1. PSAVE and ENCRYPT now properly test for ERROR 19 before writing any data to the file.
2. An ERROR 33 on CALL will no longer hang the system.
3. The START directive now works properly in the RUN mode.
4. MERGE of a multiple line statement with blanks extending from one line to the next now works properly.
5. The EDIT directive will now prevent editing the current statement when the result can not be executed predictably. An ERROR 27 is issued much the same as attempting to edit a statement with an active FOR or GOSUB directive. Basically, the portion of a statement which has already been executed may not be modified. This includes an ESCAPE directive. This change will prevent ERROR -1, ERROR 49 and various hang situations which have occurred in the past.
6. A problem where the key pointer was in the wrong position after executing REMOVEs at the end of the file has been corrected.
7. A problem with the EPT calculation has been corrected. The EPT(87819.58) now returns 5.
8. A task waiting for XON can now be released.
9. A printer configured with a model code of 0000 will default to the 0300 model code.
10. The international printer translation will no longer hang on smart boards.

11. A task on a dumb board may be released without hanging the system.
12. A public program which is ERASEd while it is ADDRed will now properly be DROPPed.
13. ERASE has been corrected to eliminate unnecessary disk activity.
14. The console hang that occurred when using Ctrl/S-Ctrl/Q has been fixed.
15. False syntax errors generated by LIBRARIAN and SCRIPT-IV have been corrected.
16. Multiple occurrences of input-verification with LEN = option will now list correctly.

Upgrade to V6.6.2 from V6.6.1

Hardware Specific Notes

IBM PS/2 Computers Only

- External 5-1/4" Diskette Drive: Current versions of Thoroughbred/OS will not format diskettes on the IBM PS/2 external 5-1/4" 360KB diskette drives. However, once diskettes have been formatted under TOS on another system using a 360KB drive, the PS/2 versions of TOS will read and write these diskettes with no problem.
- NOTE: Diskettes formatted and written at 360KB on a 1.2MB drive may or may not be readable (or writeable) on ANY 360KB drive. This is a hardware restriction which is unrelated to TOS capabilities.
- The primary removable storage medium for the PS/2 computer family is the 3-1/2" diskette drive. The intended function of the external 5-1/4" drive is to migrate software to a PS/2 system from another system, not as a back-up/restore medium for the PS/2. The 3-1/2" format is fully supported by TOS on the PS/2.
- Software Warning: The maximum PS/2 memory useable under TOS is 576 kilobytes. Configuring more memory may hang the system.

Micro Five S5000 Computers Only

- On the Micro Five 5000, Thoroughbred/OS supports only the Western Digital ST-506 hard disk controller, which is IBM PC-AT compatible. TOS does not support the SCSI-format hard disk controller on this computer. Either of these controllers may be furnished with the Micro Five 5000 -- and the correct controller must be installed if the system is to run TOS.

Software Enhancements & Changes

1. Problems have been fixed to enable the user to recover from a bad or missing Thoroughbred PASSPORT.
 - a. A bad or missing serial port interface no longer hangs the system on checking the PASSPORT because of a new feature of timeout on serial output.
 - b. If the number of authorized (non-PASSPORT) boots used is out of range (more than 7), the logic is reset to allow a system boot with an override Authorization Code.

- c. The problem that the system will not boot if port 3 (port 2 on single or 3-user system) is not configured, is resolved. Now the user is allowed to boot under this condition with a default 3-user configuration. The user must then configure port 3 (port 2 on a single or 3-user system) and reboot, in order to use the full TOS port configuration.
2. The TOS clock handler is now invoked by BIOS upon booting TOS so the machine can go directly into high speed mode on certain computers (e.g. Multitech 386).
3. A problem associated with the NCR PC-8 floppy drive when switching between 360KB and 1.2 mb formats has been fixed.

Utility Enhancements & Changes

1. The *3PSD utility now allows the copying of the Thoroughbred Operating System diskette. This utility has also been corrected to allow the copying and formatting of multiple diskettes without errors.
2. The *1PSD1 and INSTAL41 utilities have been modified to allow bad track sparing with ESDI hard disks, provided the hardware supports ESDI disks. These utilities also no longer use the existing bad track table when sparing is done on a system which already has TOS installed. Also, the HSA function is now correct when bad track sparing is not run.
3. *PARTDSK now calculates correctly if a <Return> for default values is pressed at various prompts.
4. The *WPSD utility has been changed to prevent a possible error 33 during the verification phase of a tape backup.
5. Under certain circumstances, the DOSLINK utility would add an additional sector to a TOS file moved to DOS. This has been corrected.
6. *REL is a new utility program that displays the release date of the utilities supplied with a TOS release.
7. The *VPSD utility has been changed to correct problems with (a) labeling diskettes, (b) calculating the number of required diskettes for backup, and (c) locating the floppy directory when a hard disk directory has been disabled. Note that READ-caching for a floppy drive must be turned on in order for *VPSD to function correctly.
8. The following prompts and error messages have been added to *VPSD. The page number refers to the TOS Reference Manual.

Backup

Page 9-80 before the prompt, 'SELECT DISK(S) TO BE BACKUD UP:'

SELECT THE REMOVABLE DISK MEDIA TO BE USED FOR BACKUP.

ENTER: 'F' OR 'CR' TO USE FLOPPY DISKETTE OR 'R' TO USE REMOV-
ABLE HARD DISK.

ENTER THE UNIT NUMBER OF YOUR FLOPPY DISK (F0 OR F1)

Asks only if 2 or more floppy drives are configured. If 'F' or 'CR' was selected, the system will ask for the unit number of the floppy drive.

Pages 9-90/91 additional error messages.

FILENAMES DO NOT MATCH, CR TO CONTINUE, CTL2 TO SKIP FILE:
File name information on the hard disk directory differs from the filename in the file.

OF SECTORS DO NOT MATCH, CR TO CONTINUE, CTRL2 TO SKIP
FILE:

Number of sectors on the hard disk directory differ from the number of sectors in the file.

F# NOT FOUND 'CR' TO CONTINUE

The system did not recognize the floppy directory specified.

Restore

Page 9-80 before the prompt, 'RESTORE COMPLETE BACKUP Y/N?'

SELECT THE REMOVABLE DISK MEDIA TO BE USED FOR RESTORE.

ENTER: 'F' OR 'CR' TO USE FLOPPY DISKETTE OR 'R' TO USE REMOV-
ABLE HARD DISK.

ENTER THE UNIT NUMBER OF YOUR FLOPPY DISK (F0 OR F1):

Asks only if 2 or more floppy drives are configured. If 'F' or 'CR' was selected, the system will ask for the unit number of the floppy drive.

Page 9-90 following the prompt, 'FILE BYPASSED'

MOUNT FLOPPY VOLUME #, CR TO CONTINUE

Insert the floppy needed to restore more files.

NO MORE FILES FOUND TO RESTORE

Appears only when a partial restore has been selected and a range of files or one individual file has been restored fully. It will also be displayed if no files at all could be located on the diskette.

Pages 9-90/91 additional error messages

LABEL DOES NOT MATCH, SHOULD BE VOL # OF #, CR TO CONTINUE

Floppy vol # inserted was out of sequence, insert the right floppy and <Return>.

BEGINNING OF FILE EXISTS ON PRIOR DISKETTE, CR TO SKIP FILE
The file spans 2 or more diskettes with the start of the file on a previous diskette. The system will skip this file and continue restoring.

F # NOT FOUND 'CR' TO CONTINUE

System did not recognize floppy directory specified.

FILENAMES DO NOT MATCH, CR TO CONTINUE, CTL2 TO SKIP FILE:
File name information on the floppy directory differs from the file name in the file.

OF SECTORS DO NOT MATCH, CR TO CONTINUE, CTL2 TO SKIP FILE:

Number of sectors on the floppy directory differ from the number of sectors in the file.

RECORD SIZES DO NOT MATCH, CR TO CONTINUE, CTL2 TO SKIP FILE:

Record size information on the floppy directory differs from the record size in the file.

KEY SIZES DO NOT MATCH, CR TO CONTINUE, CTL2 TO SKIP FILE:

The key size information on the floppy directory differs from the key size in the file.

STARTING FILE NOT FOUND ON VOLUME #, CR TO CONTINUE:

When only a partial restore is selected, the starting file of the range is on a subsequent diskette.

Known Problems

1. ***VPSD Utility:** Previous versions attempted to bypass bad tracks on floppy diskettes during backup. This algorithm was not successful in all cases.

*VPSD has now been modified to abort a backup when it encounters a floppy diskette error. Therefore, we strongly recommend that floppy diskettes be formatted and verified using the *1PSD Utility immediately before their use with *VPSD.

2. **Compaq 386 ONLY:** Using the floppy as a logical device (attached using *UPSD) results in very slow floppy performance. Initializing floppy diskettes with *1PSD may be slightly slower than previous releases. The performance of *VSPD however, should not be affected by this problem.
3. **IBM PS/2 ONLY:** The maximum memory usable under TOS is 576 kilobytes. Configuring more memory may hang the system.
4. **PSAVE and ENCRYPT Directives:** Both directives result in increased disk storage requirements of 16 bytes over a SAVED program. The PSZ function allocates disk space in 256-byte pages for programs, based on their size before PSAVE or ENCRYPT. If you PSAVE or ENCRYPT a program whose size (PSZ) is within 16 bytes less than a multiple of 256, you will lose part of your program on disk (the program in memory should not be affected). PSAVE will get Error 103 on the PSAVE and Error 19 on LOAD or RUN. ENCRYPT will generate no errors on the ENCRYPT, but will get Error 19 on LOAD or RUN.

SOLUTIONS: For PSAVE, use the following syntax:

PSAVE "prog-id", PSZ + 16, disk-no, sec-no, PWD = "pswd"

For ENCRYPT, when encrypting from "prog-1" to "prog-2", first issue a PROGRAM directive to define the needed space:

LOAD "prog-1"
PROGRAM "prog-2", PSZ + 16, disk-no, sec-no
ENCRYPT "prog-1", "prog-2", PWD = "pswd"

For ENCRYPT where the source program is the same as the target program (an ENCRYPT 'in-place'), load the program into memory, erase it from disk, and then PSAVE it back to disk using a sufficient program size. For example:


```
LOAD "prog-id"  
ERASE "prog-id"  
PSAVE "prog-id", PSZ + 16, disk-no, sec-no, PWD = "pswd"
```

5. PSAVE and ENCRYPT (Public Programs): Public Programs must not be PSAVEd or ENCRYPTed. They will not execute properly when invoked with either CALL or RUN.
6. If a RELEASE command is issued to either a user task or a ghost task that is running a Public Program, the task will return an Error 38.
7. Record Buffer Area Overflow: If multiple OPENs are performed within one memory bank and the additive record sizes exceed the total allocated Record Buffer Area, the system may hang. (Normal operation would be to return an Error 32.) The following environment and programs will periodically cause a hang condition, and result in various other error conditions, depending upon the length of the WAITs and whether T0 is in memory bank 0 when this is executed or in bank 3 already. On occasion, this program will cause the program "SCREEN" to display on T0, with the TSK(0) indicating that T1 was inactive. For example:

```
DIRECT "8KFILE",10,10,8704,0,0    [with data in 3 records]
```

Program BIGSTART:

```
0005 WAIT 5  
0010 START 1,BNK = 3,"BIGREAD","T2"  
0020 PRINT "T2 WITH BIGREAD"  
0025 WAIT 5  
0030 START 1,BNK = 3,"SCREEN","T1"  
0040 PRINT "T1 WITH SCREEN"  
0045 WAIT 5  
0050 PRINT "HERE I GO"  
0060 START 1,BNK = 3,"BIGREAD"
```

Program BIGREAD:

```
0010 OPEN (1) "8KFILE"  
0020 EXTRACT RECORD (1) A$  
0030 WRITE RECORD (1) A$  
0040 PRINT X,; LET X = X + 1  
0050 CLOSE (1); GOTO 0010
```

Program SCREEN:

```
0010 DIM A$(22*80,"-")
0020 FOR X = 1 TO 100
0030 PRINT A$
0040 PRINT 'CS'
0050 NEXT X
```

Therefore, the user must avoid the possibility of starting multiple tasks in a single bank, which will require more Record Buffer Area than is defined.

8. Attempting to WRITE with KEY = to a full file will generate an Error 2 as it should, and KEY will point to the next sequential KEY value had there been space to insert this record, but the IND value after the Error 2 is not always correct. If the attempt to WRITE was beyond the last KEY value in the file, both KEY and IND will return Error 2, which is correct. If the attempt to WRITE used a KEY that was less than the last KEY of the file, IND will return -1 (negative one).

NOTE: Use of the IND function in dealing with KEYed access files is not advisable and is less efficient than use of the KEY function.

9. Inability to RELEASE Active Tasks: Under some circumstances, a RELEASE of an active task from the console (T0) can either hang the system or create a large volume of output to print on the RELEASEd task's screen. The data printed appears to be of memory dump quality.

The user or system administrator should ensure that a task is inactive before RELEASEing it, by going to the task's terminal and suspending operations.

10. Invalid Write Protect Returned During *WPSD Backup: If an attempt is made to backup the system with a write-protected tape, the system will respond that the tape cannot be initialized and should be changed. If a non-write-protected tape is then placed in the drive, a message stating that the tape is SAFE (write-protected) appears even though the tape is not SAFE. If this condition is encountered, the user should exit to the main menu of *WPSD and retry the backup again with the non-write-protected tape.
11. EXTRACT and IND do not function together properly. This relates to the use of IND with Keyed access files. If the user defines a Direct file, WRITES to IND = n, EXTRACTs using IND = n, then WRITES with no IND or KEY using new data, the new data goes nowhere and the IND = n record still contains the data from the original WRITE. The following illustrates the environment:

```
DIRECT "FIX06",8,10,10,0,0
OPEN (1) "FIX06"
WRITE (1,IND=3) "TEST1"
READ (1,IND=3) A$           [A$="TEST1"]
EXTRACT (1,IND=3) A$        [A$="TEST1"]
WRITE (1) "TEST2"
READ (1,IND=3) A$           [A$="TEST1"- should be "TEST2"]
```

NOTE: Use of the IND function when dealing with keyed access files is not advisable.

12. Syntax check of certain hexadecimal constants fails: Each of the following LET statements should generate a syntax error (Error 20). Neither one does, and the results in A\$ are shown:

```
LET A$=$FFFF FFFF$         [A$=$FFFF0FFF$]
LET A$=$ $FFFFFFFF$         [A$=$0FFFFFFFF$]
```

13. KEY function returns the wrong data in some cases at end of file. Two tasks are dealing with the same file. Each task is pointing at the last record in the file (both KEY functions indicate the last key in the file). One task removes the last record (last key). The second task performs a KEY function and gets the first key of the file, where it should have gotten an Error 2. (This same task, if it performs another KEY function without any other I/O operation, will get an Error 2, which it should have received on the previous operation.)

14. Null Records can be written in two formats, but generate different results: The following two commands generate the results shown:

```
WRITE (1) ""               Writes a record containing $8A$ in the first byte.
WRITE (1)                  Writes a null record.
```


Upgrade to V6.6.1 from V6.6

Software Warning

1. A problem with encryption is resolved in 6.6.1. Programs encrypted using the ENCRYPT or PSAVE directive under earlier versions will not operate under 6.6.1. These programs must be decrypted before upgrading to 6.6.1.

Utility Enhancements & Changes

1. *1PSD(INSTAL4) is changed to have the capability to format 1.44Mb/720K floppy diskettes. *1PSD no longer displays the "VERIFYING FORMAT: " message.
2. *1PSD now performs faster when used to format a 1.2Mb floppy diskette.
3. *3PSD is changed to accommodate the 720K and the 1.44Mb diskette formats. This utility, while copying to disk, automatically detects which format is being copied and formats and copies the destination diskette accordingly.

Software Enhancement

1. READ RECORD with a substring reference as the receiving variable is now supported.

Upgrade to V6.6 from V6.5.3

Software Warning

1. The Thoroughbred 2-Mbyte 10MHz Memory Board has been tested on an 8MHz IBM PC-AT and some compatible microcomputers. The memory board is not designed to operate on microcomputers faster than 8MHz.
2. It is very important that you **DO NOT CHANGE** the default configuration for Memory Banks and for any of the System Options (especially the Sector Multiple). Improper operation may result if these parameters are changed.
3. The current version will operate with either the monochrome monitor or Color Graphics Adapter (CGA) but will not run with an Extended Graphics Adapter (EGA) option.
4. In the *NPSD Utility, Serial Port #3 should not be deleted; otherwise, booting problems may occur. Also, the first three ports (configured for the COM1 Remote Panel) must have devices configured on them (even if the devices are not present); otherwise the Thoroughbred PASSPORT may not be recognized.
5. In the *NPSD Utility, the disk directories must specify at least one device as being cached. We recommend that the caching for floppy drives be Read-only; and for hard disk drives, Read-and-Write (or Read-only if there is a concern about data loss due to power interruptions).
6. TOS V6.6 utilities are incompatible with earlier versions of TOS. Do not substitute TOS utilities between V6.6 and any earlier version!
7. Although the Loading Procedure implies that TOS will work with different hardware configurations, Concept Omega has not at this time tested TOS on the PC-AT with Maynard Disk Drives.

Known Problems

1. Serial Files currently are limited to a 1500-byte record size; sizes larger than this may cause an Error 1 (End of Record).

Until the problem is fixed, the following method can be used as a temporary solution that will extend the 1500-byte limit by artificially expanding the Record Buffer:

Dimension a variable to the maximum desired record size and use it in the GET directive. Example: DIM A\$(5000); GET 0,0,A\$

Note that Serial Files will also produce an Error 1 on a standard End-of-Record error condition (as expected).

Software Enhancements & Changes

1. Thoroughbred PASSPORT and SSN: Implementation of the SSN feature using the Thoroughbred PASSPORT Serial Number, verifying that the proper class of T-PASSPORT is being used and that the number of users authorized by the T-PASSPORT is not exceeded in Thoroughbred BASIC. The SSN can now be used by our dealers for use with PSAVE and ENCRYPTed save to prevent unauthorized use of their software. Dealers will be able to permanently link their software products to a specific class of system, number of users, and/or serial number based on the T-PASSPORT's SSN. TOS requires the "R" type security box, except the TOS 1-user version which requires the "I" type box.

The new format of the 9-digit SSN is as follows:

Bytes	Description	
1-2	System Classification:	
	00 = DOS	40 = Mini Computers
	10 = PC XT, AT & Compatibles	50 = Supermini Computers
	20 = Micro Computers	60 = Mainframes
	30 = Supermicro Computers	
3-4	User Classification (* = TOS only)	
	05 = Single User	45 = 40 Users*
	10 = 3 Users*	50 = 48 Users*
	15 = 6 Users*	55 = 56 Users*
	20 = 8 Users	60 = 64 Users
	25 = 10 Users*	65 = 128 Users
	30 = 16 Users	70 = 256 Users
	35 = 24 Users*	75 = 512 Users
	40 = 32 Users	80 = 1024 Users
5-9	Unique Number (00000 to 99999) within User/System Class	

2. Extended Memory: TOS 6.6 incorporates a 2 MByte Memory Board within the TOS 10-User and 16-User TURBO Systems to increase system memory beyond 640K. This additional 2 meg of memory can be used for user banks or as cache memory. Configuration of Memory in the *NPSD Utility has been modified. Also the Compiler and Lister will now always be resident; the

*NPSD System Options selection for Resident Compiler & Lister has been removed.

3. Allow CPU processing during disk seeks: To speed up system performance, the TOS 6.6 now continues with CPU processes while waiting for disk I/O to complete.
4. Expanded Bad Track Table: The Bad Track Table for the hard disk has been expanded to allow 79 entries. The table is initialized during installation and using the *IPSD Utility.
5. Support of AT internal clock: TOS 6.6 now utilizes the AT system clock which is initialized with the SETUP Utility under PC-DOS.
- 6.. Increased number of ghost tasks: The maximum number of ghost tasks TOS 6.6 supports has been increased from 4 to 10. Valid ghost task identifiers are G0 - G9.

Utility Changes & Enhancements

1. *APSD, *DPSD, *EPSD - These utilities have been modified to support Serial files and encrypted programs.
2. *FPSD - Now displays correct file information for Serial files. It also identifies an encrypted program.
3. *IPSD - Provides the ability to examine Serial files.
4. *LPSPD - Supports the listing of Serial file types with the necessary associated data. Displays total sector and total bytes information for the listed files.
5. *NPSD - There have been numerous changes to this utility. The screens and prompts for SERIAL PORT and MEMORY configuration have changed drastically reflecting 16 user and extended memory support. It is strongly recommended that you DO NOT change the default configurations of memory and serial ports (except to add serial printers). To help prevent possible invalid configurations from being saved, *NPSD now calculates and displays memory requirements for OS tables. It also now does a better job of indicating when an invalid configuration has been created.
6. *OPSD - This utility has been changed to support the creation of Serial files. It also now provides an option to null (with hex 00) an Index file after the file has been created.
7. *PPSD - This utility checks and corrects "END" line as above. It also does NOT change any references made to line 0000 (SETESC 0, etc.)

8. *QPSD - As this utility now processes files, it checks for a proper "END" line at 10000 and makes the correction if necessary. The utility also, in search and replace mode, checks for proper DEF table layout and makes corrections if necessary.
9. *SETDSK - This utility now supports the "OTHER" hard disk option that was featured in TOS 6.5.2.
10. *SPSD - Allows the transfer of Serial files and encrypted programs. It also provides an option to erase the file(s) from the source directory and LOCKS files as they are copied to prevent other users from accessing files as they are being copied.
11. *TPSD - This utility has been changed to support Serial files.
12. *VPSD - The Diskette Backup and Restore Utility has been revised to support Serial files and encrypted programs. The *VBU option is now exclusive. If this option is selected then you cannot select other disk directories to backup. This was done to eliminate all sequence related problems on restore.
13. *WPSD - The Tape Backup/Restore Utility has been modified to support Serial files and encrypted programs. Other enhancements include verification of backup tape, accomodating backups when bad files exist on disk (Error 103), and lessening the time required to perform a Restore.

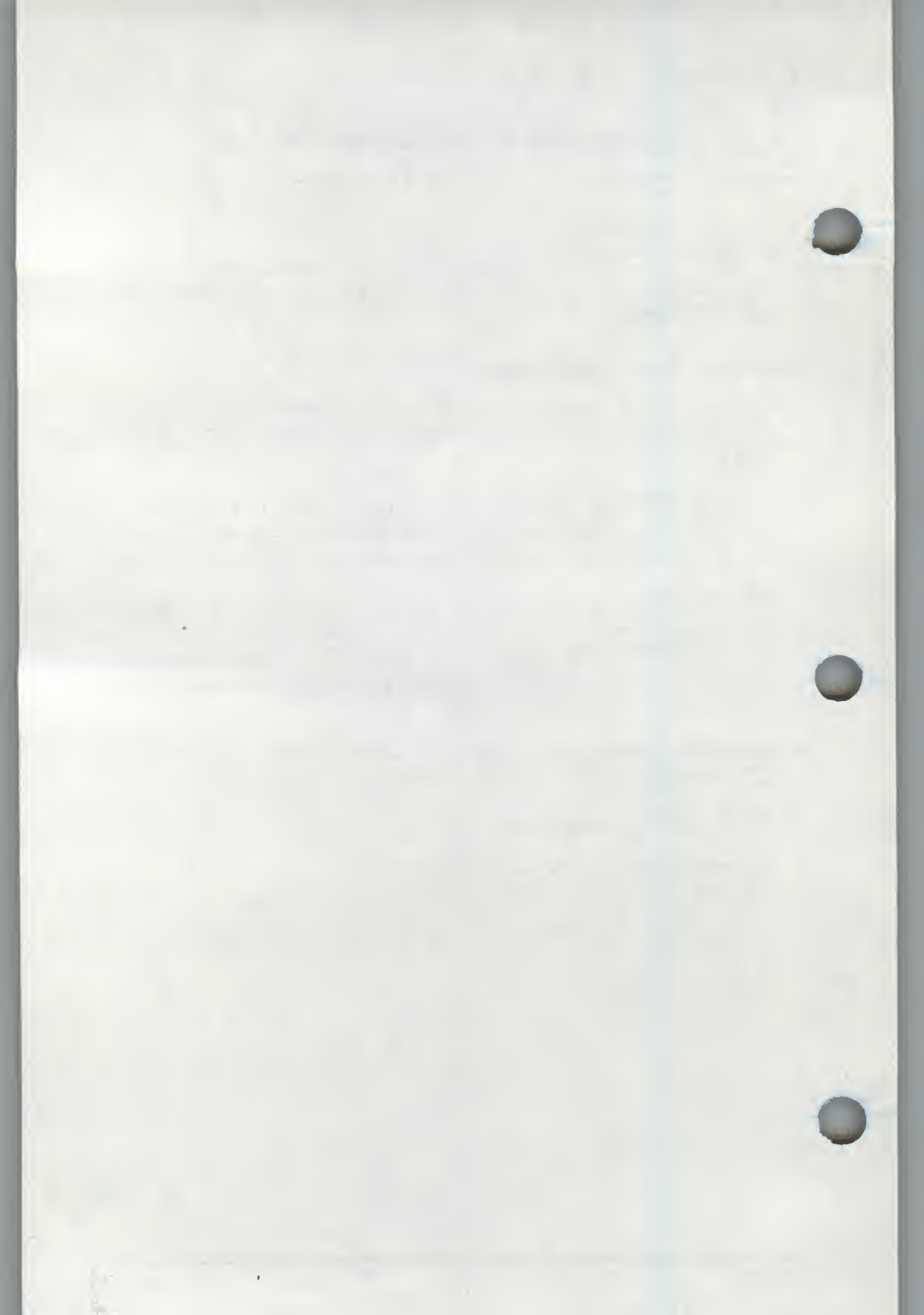
Upgrade to V6.5.3 from V6.5.2

Software Warning

1. TOS V6.5.3 Utilities are incompatible with earlier versions of TOS. DO NOT SUBSTITUTE TOS UTILITIES BETWEEN V6.5.3 AND ANY EARLIER VERSION!

Software Enhancements

1. Starting with TOS Version 6.5.1 for the PC-AT, TOS fully supports the High-Capacity Diskette Drive on the PC-AT using both the 1.2Mb and 360Kb formats.
2. Starting with TOS Version 6.5.3 for the PC-AT, TOS supports the Serial/Parallel card(s); the PC-XT Asynchronous Communications Adapters (Serial Cards) are no longer a requirement for TOS on the PC-AT.
3. Communication Enhancements: (a) Ability to transmit \$FF\$ and \$00\$. (b) Ability to communicate at 9600 baud in raw and buffered modes. (c) Raw mode option for \$0D\$ delimiter. (d) XON/XOFF protocol support. (e) Direct 3-user port initialization (bypassing BIOS). (f) Corrected bug with CR/LF echo after input termination.
4. Support was added for IBM AT serial/parallel cards. Now both the DB25 XT serial cards and the DB9 AT serial cards are supported.
5. CTL-S and CTL-Q now work on T0.
6. The internal buffer has been expanded to 2K.



*Thoroughbred/OS*TM

Thoroughbred/OS™

COPYRIGHT (C) 1987 CONCEPT OMEGA CORPORATION
All rights reserved.



This documentation may not be reproduced or modified in any way, without the express written permission of:

Thoroughbred® Concept Omega® Corporation

19 Schoolhouse Road o PO Box 6712 o Somerset, NJ 08875-6712
Phone: 201-560-1377 o Telex: 910-3808-394 o Fax: 201-722-7958

Thoroughbred/OS, Thoroughbred, Thoroughbred PASSPORT, and Concept Omega are trademarks of Concept Omega Corporation. PC-DOS, IBM, PC-XT, and PC-AT are trademarks of International Business Machines Corporation. MS-DOS is a trademark of MicroSoft Corporation.

TABLE OF CONTENTS

SYSTEM PROCEDURES	i
CHAPTER ONE—INTRODUCTION	1-1
Manual Organization and Conventions	1-2
CHAPTER TWO—DATA REPRESENTATION, HANDLING AND PROCESSING	2-1
Introduction	2-1
Data Representation	2-2
Numeric Data	2-2
Alphanumeric Data	2-4
Data Handling and Processing	2-5
Expressions	2-5
Functions	2-7
System and Task Variables and Functions	2-9
Assignments	2-11
Internal Representation	2-11
Program Statements as Data	2-12
Functions and Directives	2-13
CHAPTER THREE—SYSTEM DIRECTIVES	3-1
Tasks	3-1
Programs	3-2
Program Mode Processing	3-2
Console Mode Processing	3-3
Escape Processing	3-3
Directives	3-4
CHAPTER FOUR—PROGRAM CONTROL	4-1
Program Processing	4-1
Compound Statements	4-1
Return Address Stack	4-2
Directives	4-5
CHAPTER FIVE—DISK AND FILE OPERATIONS	5-1
Disks	5-1
Files	5-2
File Types	5-2
File Creation	5-3
File Accessing	5-4
Directives	5-5

CHAPTER SIX—INPUT/OUTPUT PROCESSING	6-1
Input/Output	6-1
I/O Directive and Option Capabilities	6-2
Record Specification	6-2
Error Branching	6-3
Input/Output List	6-4
Code Conversion	6-4
Miscellaneous I/O Options	6-4
Input Verification	6-5
Data Positioning	6-8
Mnemonic Device Control	6-10
Output Formatting (Masking)	6-13
Special Character and Code Representation	6-15
Directives	6-17
 CHAPTER SEVEN—PUBLIC PROGRAMS	7-1
Public Program Process	7-1
Restrictions on Public Programs	7-2
Transfer of Values	7-2
Directives	7-4
 CHAPTER EIGHT—ERROR PROCESSING	8-1
Syntax Errors	8-1
Execution Errors	8-1
Error Trapping and Processing Capabilities	8-2
Untrapped Errors	8-2
Error Task Variable	8-3
Error Code Listing	8-3
 CHAPTER NINE—THOROUGHbred/OS UTILITIES	9-1
Control/Function Keys	9-2
Utilities	9-3
 APPENDIX A—ASCII CODES	A-1
 APPENDIX B—VFU LOADING	B-1
 INDEX	I-1

CHAPTER ONE—INTRODUCTION

Thoroughbred/OS is a multi-user BASIC Operating System with an enhanced Business BASIC programming language included. Thoroughbred/OS was developed to provide greater power and efficiency in solving business application problems on today's new generation of business computers.

Thoroughbred/OS supports up to 16 users and is interpretive, allowing immediate reaction to commands as they are entered. This feature also includes immediate response to syntax and execution errors. Thoroughbred/OS is also compiled to decrease storage space and to increase processing speed.

Some of the specific features of Thoroughbred/OS are:

- Versatile data representation and handling
- Extensive system commands
- Adaptable program processing
- Disk and file capabilities
- Input and output processing
- Support of efficient Public programming features
- Extensive error handling and processing capabilities
- Comprehensive collection of supporting Utility programs
- Support of Ghost or Background processes

This manual is organized as follows:

Chapter 2—Data Representation, Handling and Processing

Chapter 3—System Directives

Chapter 4—Program Control

Chapter 5—Disk and File Operations

Chapter 6—Input/Output Processing

Chapter 7—Public Programs

Chapter 8—Error Processing

Chapter 9—Thoroughbred/OS Utilities

Appendix A—ASCII Code Representations

Appendix B—VFU Loading

The syntax conventions used in this manual are:

parentheses () surround required parameters. Parentheses are entered with the parameters they surround.

brackets [] surround optional parameters. Brackets are not entered with the parameters they surround, except for the Editing directive.

numeric-value is a constant, variable, or expression that is a numeric value.

string-value is a constant, variable or expression that is a string value.

The following Control sequences have the described effects:

CTL B or CTL b terminates processing of BASIC by interrupting any current process and returns control to the operating system.

CTL S or CTL s interrupts and halts terminal display, as when listing or tracing statements.

CTL Q or CTL q resumes the display halted by the CTL S response.

NOTE: All hexadecimal and binary representations assume the use of the ASCII code in which bit 8 is set to 0.

CHAPTER TWO—DATA REPRESENTATION, HANDLING AND PROCESSING

Any computer activity, such as executing a program or processing an interaction between a user and the computer, deals with the representation, handling, and processing of information or data.

For a successful interchange between the computer and the user, certain conventions and procedures must be used. The conventions and procedures used by Thoroughbred/OS form the extensive and facile capabilities of this programming language, and they include:

- Provision for constant and variable numeric and alphanumeric values.
- Variable and extended numeric precision and floating point representation capability.
- A complete series of arithmetic and relational operators.
- Pre-programmed and user-programmable functions.
- Functions and variables that return System and Task information.
- Handling of hexadecimal values and operations on the binary representation of data.
- Handling of the listed and compiled forms of BASIC statements.

This chapter describes specific details regarding numeric and string data, the functions, variables and directives provided, and other features related to data representation, handling and processing.

DATA REPRESENTATION

BASIC can handle two types of data: numeric and alphanumeric (also referred to as "string" data). Each type may occur in either a constant or variable form.

The following sections define and describe the two types of data and the forms they use.

Numeric Data

Numeric data is represented by either positive or negative numerals. A plus sign (+) is optional for positive numeric data; a minus sign (–) must precede negative numeric data.

Thoroughbred/OS allows three forms of numeric data—Integer Format, Fixed-Point Format, and Floating-Point Format.

Integer Format

Integer format consists of whole numbers. Some examples of numeric data in integer format are:

3
– 47
236

Fixed-Point Format

Fixed-point format consists of one or more numbers followed by a decimal point. The decimal point may appear before, after, or among the numbers. Some examples of numeric data in fixed-point format are:

– 3.1
.23
7.
– 345.69

Floating-Point Format

Floating-point format specifies a value that is multiplied by 10 (E) raised to the indicated exponential power. Some examples of numeric data in floating-point format are:

2E3 which says: 2 times 10 to the third power (= 2000)
.99E – 4 which says: .99 times 10 to the negative fourth power
 (= .000099)
– .99E5 which says: – .99 times 10 to the fifth power
 (= – 99,000)

NOTES

- A numeric value can have a maximum of 14 significant digits, plus a decimal point. Values with more than 14 digits may be expressed in floating-point format.
- Maximum and minimum values that can be handled in fixed- and floating-point format are:

99999999999999 to - 99999999999999

.99999999999999E + 141 to - .99999999999999E - 113

Numeric Constants

A numeric constant is a number whose value is never changed during the execution of a program. Thus, the integer one (1) is a constant in the following BASIC statement:

```
LET X = X + 1
```

Numeric Variables

A numeric variable refers to numeric data whose value is subject to change during the execution of a program, and is therefore represented by a symbolic name.

Numeric variables are denoted by a single alphabetic character, or an alphabetic character followed by a single numeric character. Some examples of numeric variable names are:

D

N8

Numeric Subscripts

Subscripted numeric variables specify elements in a numeric array and consist of any valid numeric variable followed by a subscript in parentheses. Some examples of numeric subscripts are:

A(4)

C(2,3)

K(M,N)

B1(6)

Alphanumeric Data

The other form of data allowed by Thoroughbred/OS is alphanumeric data, also referred to as string data. String data has a character value consisting of any combination of numbers, letters, and special characters. Legal characters include all alphabetic, numeric, special characters and control characters provided for in the representational scheme of ASCII (see Appendix A for a list of ASCII code representations).

String data may be words, sentences, or arrangements of characters, enclosed in delimiters. ASCII string constants are delineated by enclosing quotes (" "), while hexadecimal values are delineated by dollar signs (\$ \$). Some examples of string data are:

"ABCD"

"1234"

"A#2\$"

"ENTER INVENTORY NUMBER"

NOTES

- The null string (a string containing no characters) is represented by a pair of quotes with no intervening spaces ("").
- The maximum size of a string is limited only by memory considerations.

Character Variables

A character variable is string data whose value is subject to change during the execution of a program.

String variable names consist of a single alphabetic character from A to Z, optionally followed by a single numeral from 0 to 9, and are always followed by a dollar sign (\$) to distinguish them as string variable names. Some examples of string variable names are:

G\$

M7\$

String Subscripts

Subscripting of string variables is used to specify a substring that is made up of a portion of the original string. The following format is used:

variable-name (1st-position [,length])

where:

1st-position indicates at what position to begin the substring

length indicates how long the substring is to be

For example: If A\$ = "STRING", A\$(2,3) specifies the substring "TRI". If length is not specified, the substring is evaluated to the end of the string; i.e., A\$(3) specifies the substring "RING".

DATA HANDLING AND PROCESSING

Data handling and processing is accomplished by a large number of BASIC language features. Arithmetic operations and a large number of functions are used to manipulate and calculate values. Special predefined functions and variables allow easy access and processing of commonly used System and Task status information. BASIC also provides the capability of handling the assignment of values on a conditional basis. Features are also provided that allow for special forms of numeric processing. The following sections describe these capabilities.

Expressions

An expression is a series of data elements consisting of constants, variables, functions, or expressions that interact with the appropriate operators to form a new value. Expressions may contain either numeric or string values.

Numeric Expressions

Numeric expressions contain numeric data elements that are related with a series of arithmetic operators. The arithmetic operators provided are:

SYMBOL	MEANING
+	addition
-	subtraction
*	multiplication
/	division
^ or **	exponentiation

Some examples of numeric expressions are:

$A * 5$
 $B + C \quad 2 * 6$
 $(3 * 4) / 2 - D$

Evaluation of Numeric Expressions

Numeric expressions are evaluated according to the operator's priority. Operations with higher priorities are performed first. Those at the same priority level are performed as they are encountered, left to right. Operations in an expression are processed in the following order:

- () expressions within parentheses indicate grouping and are evaluated as a single value
- ^ or ** exponentiation
- *, / multiplication and division
- +, - addition and subtraction

The following is an example of the evaluation of arithmetic operations:

$4 + 4 / 2 * 5 \wedge 3$

The evaluation process follows the sequence:

1. $5 \wedge 3$ (result is 125)
2. $4/2$ (result is 2)
3. $2 * 125$ (result is 250)
4. $4 + 250$ (result is 254)

The answer to the problem is 254.

String Expressions

A string expression contains string data elements that interact with the string operator, a plus sign (+), to form a new string value. Only one operation is provided for string expressions: concatenation—the combining together of a sequence of characters. Some examples of string expressions are:

A\$ + B\$
"UNITED" + "STATES"
"UNITED" + A5\$

Numeric and String Functions

Functions are expressions or operations on one or more values that return a new value. The expression or operation is often a complicated or common one that is given a symbolic name in order to conserve space and time during program coding. The actual operation performed is referred to only by its function name, and is therefore 'invisible' to the programmer. Many predefined internal functions are provided, along with the capability to define new functions.

Numeric Functions

The types of numeric functions provided are:

Arithmetic	process arithmetic operations
Conversion	return a numeric result from a string
Programmable	allow the defining of a new function

The numeric functions covered in this chapter are:

ARITHMETIC

ABS	Absolute Value
EPT	Exponent Value
FPT	Fractional Part
INT	Integer Value
MOD	Modulo Function
SGN	Sign Value

CONVERSION

ASC	ASCII Function
DEC	Decimal Function
LEN	Length Function
NUM	Numeric Function
PCK	Pack Function
POS	Position Function
UPK	Unpack Function

The following are conversion-related functions that operate on a numeric value and return a string value:

BIN	Binary Function
CHR	Character Function
STR	String Function

PROGRAMMABLE

FN	Define Programmable Function
----	------------------------------

String Functions

The types of string functions provided are:

Code Conversion	return string values from other string or numeric values
Logical/Binary	conduct logical or binary operations on the binary representation of data
Programmable	allow the defining of a new function

The string functions covered in this chapter are:

CODE CONVERSION

ATH	ASCII from Hexadecimal
CHR	Character Function
HTA	Hexadecimal from ASCII
STR	String Function

The following are conversion related functions that operate on a string value and return a numeric value:

ASC	ASCII Function
DEC	Decimal Function
LEN	Length Function
NUM	Numeric Function
POS	Position Function

LOGICAL/BINARY

AND	AND Function
BIN	Binary Function
CRC	Cyclic Redundancy Check
DEC	Decimal Function
GAP	Generate Odd Parity
HSH	Hash Function
IOR	Inclusive OR Function
LRC	Longitudinal Redundancy Check
NOT	NOT Function
XOR	Exclusive OR Function

PROGRAMMABLE

FN	Define Programmable Function
----	------------------------------

System and Task Variables and Functions

A special set of variables and functions are provided that return information on the status of a system or task, or hold the value of special system or task parameters. These functions and variables operate on and/or return both string and numeric values.

Substring operations cannot be performed directly on these functions and variables; their results must first be assigned to a variable.

The following System and Task variables and functions are covered in this chapter:

RETURNS A NUMERIC RESULT

BSZ	Bank Size (Memory) Function
CTL	Control (Function) Key Input Variable
DSZ	Data Size Variable
ERR	Error Function
HSA	Highest (Disk) Sector Function
IND	Index Function
PSZ	Program Size Variable
SSN	Serial Number Variable
SSZ	Sector Size Function
TIM	Time Function

RETURNS A STRING RESULT

DSD	Data Set Description Function
FID	File Identification Function
KEY	Key Function
PGN	Program Name Variable
PUB	Public Programs Directory Function
SYS	System Variable
TCB	Task Control Block Function
TSK	Task Memory Parameters Function
TSK(0)	Task Status (System) Function
TSM	Termination Status Message Variable

Assignments

Assignments are used to assign a value to a variable and are made using directives. The following directives for processing assignments are covered in this chapter:

LET
DIM
IF/THEN/ELSE
SETDAY
SETTIME

Special Processing Directives

The following special processing directives are covered in this chapter:

FLOATING POINT
PRECISION

Internal Representation

Thoroughbred/OS is equipped to handle the character set provided by the ASCII code representation scheme (see Appendix B for the ASCII Code Representations) along with the hexadecimal representation of string values.

Thoroughbred/OS provides a series of functions that can convert among the different code representations (Hexadecimal, Decimal, ASCII [standard]) and logical functions that operate on binary forms of data representation. These are:

CODE CONVERSION FUNCTIONS

ASC	ASCII Function
ATH	ASCII from Hexadecimal
BIN	Binary Function
CHR	Character Function
DEC	Decimal Function
HTA	Hexadecimal from ASCII

LOGICAL AND BINARY FUNCTIONS

AND	AND Function
CRC	Cyclic Redundancy Check
GAP	Generate Odd Parity
HSH	Hash Function
IOR	Inclusive OR Function
LRC	Longitudinal Redundancy Check
NOT	NOT Function
XOR	Exclusive OR Function

Program Statements as Data

BASIC statements are handled in two different forms. One form appears on the terminal, or printed listing of the program. This is the English text form of the statements. The other form is internal to the computer and consists of the compiled form of the program. Compiling conserves space when the program is stored, and time when it is executed.

BASIC has the capability of converting the compiled and text versions of a statement back and forth and manipulating them as string data. This capability can be used with the EXECUTE directive to alter and execute statements within a program. With the capability of using statements as data, program processing and logic may be altered or constructed to meet conditions as they develop within the program. The following functions, in addition to the EXECUTE directive, are provided for the manipulation of BASIC statements as data:

STATEMENT CONVERSION AND EXECUTION CAPABILITY

CPL	Compile Function
LST	List Function
PGM	Program Statement Function

Absolute Value Function

ABS

FUNCTION This function returns the absolute value of a number.

SYNTAX ABS (numeric-value)

EXAMPLES ABS (- 7.39)
returns the value 7.39

ABS (Q)
if Q = 22, returns the value 22

ABS (2 * C)
if C = 3, returns the value 12

Z = 12 + ABS (Q)
if Q = - 22, assigns Z the value 34

FUNCTION This function converts a string containing hexadecimal code characters into its equivalent standard ASCII character representation.

SYNTAX ATH (string-value)

where:

string-value is a string containing hexadecimal code characters

- DESCRIPTION**
- Two characters of hexadecimal code represent a single standard character, so that the resultant string will be half as long as the input string.
 - If an odd number of characters is used in the input string, the function assumes 0 is the first character, in order to make an even-numbered string.
 - The inverse of this function is the HTA function.

EXAMPLES

ATH ("2A")
returns the string "*"

ATH (P\$)
where P\$ = "414243", returns the string "ABC"

ATH ("A43")
returns the same string as "0A43" ("line feed C") (line feed is a single control character)

ATH (\$41\$)
returns the control character, "line feed". \$41\$ represents the character "A" which is evaluated as 0A in order to obtain the result of this function

ASCII Function

ASC

FUNCTION

This function returns the decimal code of an ASCII character.

SYNTAX

ASC (string-value [,ERR = stmt-no])

where:

stmt-no is the number of the statement to branch to if an error occurs while processing this function

DESCRIPTION

- The string value specified may be of any length but only the code value of the first character will be returned.
- The inverse of this function is the CHR function.

EXAMPLES

ASC ("A")

returns the numeric value 65 (decimal code for the character A)

ASC (S\$)

if S\$ = "BUG", returns the value 66 since only the first character is evaluated

X = ASC (R\$,ERR = 8000)

if R\$ = "" (an empty or null string), an error is produced and control is given to statement 8000

FUNCTION This function returns the number of bytes of memory available in the specified memory bank.

SYNTAX BSZ (1)

DESCRIPTION

- This value is dynamic; it is constantly being recalculated as memory requirements change during processing.
- All configured memory banks may be accessed.

EXAMPLES B = BSZ (1)
if B = 2048, there are 2048 bytes of available memory remaining in memory bank 1

FUNCTION

This function converts a decimal value into its binary form and returns the ASCII characters that the value represents.

SYNTAX

BIN (numeric-value,result-length)

where:

result-length is the length in bytes of the result

DESCRIPTION

- The binary result is right-justified and truncated, or zero-filled as necessary, to fill the specified number of bytes.
- Negative values are converted to their twos-complement form.
- This function is an extension of the CHR function.

EXAMPLES

BIN (65,1)

returns the character "A" which represents the binary value 65 (01000001)

X\$ = BIN (X,1)

if X = 65, returns the same value as above and assigns it to X\$

BIN (-3,1)

returns the character "}" which represents the binary twos-complement value equal to -3 (11111101)

BIN (16706,2)

returns the characters "AB" which represent the binary value equal to 16706 (01000001 01000010)

FUNCTION This function converts a decimal character code value into its equivalent ASCII character (only one character is returned).

SYNTAX CHR (numeric-value)

where:

numeric-value is an integer value from 0 to 255 that represents a decimal code character

stmt-no specifies the number of the statement to branch to if an error occurs during the processing of this directive

DESCRIPTION

- Since there are only 128 ASCII characters, values (x) greater than 127 produce the same character as (x - 128).
- The inverse of this function is the ASC function.

EXAMPLES

CHR (x)
if X = 65, returns the character "A"

CHR (193)
returns the same character as the first example since $65 = 193 - 128$

CHR (13)
returns the character for Carriage Return

P\$ = CHR (2*X + 5)
if X = 30, assigns P\$ the value "A"

P\$ = CHR (Y)
if Y = 300, will cause an execution error (Y is out of the range 0-255)

FUNCTION This function converts an English text BASIC statement into its compiled form.

SYNTAX CPL (string-value)

where:

string-value is the text of a BASIC statement

- DESCRIPTION**
- Compiling is used to conserve space and to increase processing speed when executed.
 - If the BASIC statement string being compiled contains a syntax error, a flag is set (the third byte of the output string is set equal to \$F3\$). When an attempt is made to execute or list the output of this function, an execution error will be returned. The fourth and fifth bytes of the output string contain the position in which the error was found.
 - The BASIC statement being compiled does not need a statement number.
 - The compiled form of a BASIC statement may contain unprintable ASCII characters.
 - LST is the inverse of this function.

EXAMPLES

CPL (Q\$)

if Q\$="0100 LET X=3", returns a string whose hexadecimal form is \$006449C37BEB03F2\$

CPL "100X = 3"

returns the same string as the first example
since both statements are equivalent

Y\$ = CPL (S\$ + T\$)

if S\$ = "100" and T\$ = "X = 3", assigns Y\$ the
same value as the first two examples

Z\$ = CPL (T\$)

if T\$ = "X = 3", assigns Z\$ the value of the
previous statements, but without the portion
containing the statement number

FUNCTION This variable retains a numeric value that is set according to which Control or Function key was depressed in response to an INPUT or INPUTRECORD directive.

SYNTAX CTL

DESCRIPTION

- The value set remains intact until the next INPUT or INPUTRECORD occurs and a new Control Key is depressed.
- A BEGIN, CLEAR, END, LOAD, RESET, RUN "program-name", START or STOP directive resets the value of this variable to 0.
- The following is used in assigning a value to CTL:

Function or Control Key	Value of CTL
RETURN or LINE FEED	0
F1 or CONTROL I	1
F2 or CONTROL II	2
F3 or CONTROL III	3
F4 or CONTROL IV	4
F5 or CONTROL V	5
.	.
F _n (n = highest numbered Key)	n
SHIFT F1 (available on some terminals)	n + 1
.	.
SHIFT F _n (available on some terminals)	n + n
After entering the number of characters specified by the SIZ = option	5

EXAMPLES

X = CTL

if the F1 or CONTROL I Key was depressed, X is set equal to 1

FUNCTION This function conducts a complex logical operation on the binary form of a string value and returns a 2-byte binary value that is expressed as the ASCII characters those values represent.

SYNTAX CRC (string-value [,2-byte-string])

where:

2-byte-string is a 2-byte string that contains the result of a previous CRC operation

- DESCRIPTION**
- The result of this function is used in checking for errors in data transmission.
 - The logical operation of this function is a complex logical OR, in a predetermined pattern, on each bit on the binary form of a string.
 - This function is associative, allowing results to be accumulated for use in later CRC functions. For this capability, an optional 2-byte string, as described in the Syntax section, may be specified.

EXAMPLES CRC ("C")
returns the characters "qA" which represent the result of this operation

X\$ = CRC (\$43\$)
returns the same string as the first example and assigns it to X\$

A\$ = CRC (B\$ + C\$)
is equivalent to A\$ = CRC (B\$) followed by
A\$ = CRC (C\$,A\$)

FUNCTION This function returns information on the specified directory, device or task.

SYNTAX DSC (string-value)

where:

string-value is a device-id (e.g. LP), a logical disk number (e.g. D1), or a task-id (e.g. T0).

DESCRIPTION The information is returned in the following format:

Byte	Description
1,2	Device Designator (e.g. T0, LP...)
3	Status Byte (undefined)
4	Device Type: H = Hard disk T = Terminal F = Floppy C = Tape drive
5	Port: 0 = Terminal 3 = Floppy 6 = Hard disk 1 = Serial Printer 9 = Parallel Printer
6	Unit: 0 = Terminal 3 = Floppy 6 = Hard disk 1 = Serial Printer 9 = Parallel Printer
7-10	Directory name (model code for terminals and printers)
11	Device/Directory Status: D = Disabled R = Reserved L = Locally disabled
12-20	Unused

DESCRIPTION

- Byte 3 (Status Byte) does not appear in the returned string.
- R (Reserved) will only appear in Byte 11 for the task that issued the RESERVE.

EXAMPLES

PRINT DSD("D3")

If D3H60DEMOL is returned: D3 is the disk-id, H indicates the device is a hard disk, 6 indicates the port is to a hard disk, 0 indicates the unit is a terminal, DEMO is the name of the directory, and the L following DEMO indicates the disk is locally disabled.

LET D = DSD("T0")

PRINT D

If T0T009500 is returned: T0 is the task-id, T indicates the device is a terminal, 0 indicates the port is to a terminal, 0 indicates the unit is a terminal, and 9500 indicates the model code for Terminal T0 is 9500.

PRINT DSD("LP")

If LPP901650 is returned: LP is the device-id, P indicates the device is a printer, 9 indicates the port is to a parallel printer, 0 indicates the unit is a terminal, and 1650 indicates the model code for printer LP is 1650.

FUNCTION This variable indicates the amount of memory remaining available in the memory bank of the task that issues this function. The value returned is expressed as the number of bytes, in decimal form.

SYNTAX DSZ

DESCRIPTION • This value is constantly being recalculated as memory requirements change.

EXAMPLES D = DSZ
if D = 15216, there are 15216 bytes of memory remaining available in the memory bank that holds the user task

Day (Date) Variable

DAY

FUNCTION This variable returns an 8-byte string containing the date, in the form mm/dd/yy, currently being used as the System Date (and/or Task Date).

SYNTAX DAY

DESCRIPTION

- The date variable maintained by the system is updated every 24 hours while a date that has been set by the SETDAY directive is not.
- This variable affects the system wide value of DAY when issued by any Task.

EXAMPLES Z\$ = DAY
on April 26, 1983, Z\$ = "04/26/83"

FUNCTION This function converts a character string into the decimal ASCII code representing that string.

SYNTAX DEC (string-value)

DESCRIPTION • This function is the inverse of the BIN function.

EXAMPLES DEC ("A")
returns the value 65: the decimal code for the character "A" (01000001)

X = DEC (B\$)
if B\$ = "AB", assigns X the value 16706
(01000001 01000010)

DEC (BIN(193,1))
returns the value - 63

FUNCTION This directive allows the user to name and define a function. Once defined, it can be referred to at any point in the program by using the proper Programmable Function to call up the definition of the function. This allows an expression to be used repeatedly in a program without redefining it each time.

SYNTAX DEF FNx (variable-list) = numeric-expression
DEF FNx\$ (variable-list) = string-expression
where:

x is a single character from A to Z that specifies the name of a function

variable-list is a list of variables that receive values from the FN calling function, and are used in the definition of a function

numeric-expression is an expression that returns a numeric value

string-expression is an expression that returns a string value

DESCRIPTION

- The expressions that define a function may be formed with any number of numeric and/or string variables or constants, but are defined according to whether the result is a numeric or string value (i.e., a function that operates on a numeric value, but produces a string value is defined as a string function).
- The Programmable Functions that call up the definition of the function are:

FNx (Programmable Numeric Function) for functions that produce numeric values.

FNx\$ (Programmable String Function) for functions that produce string values.

- The variables in the variable list of this directive and in the value list of the Programmable Function may have different variable names but they must be in corresponding positions in both lists. The first variable in the Programmable Function value list sends its value to the first variable in the defining directive's variable list, etc. (i.e., if the value list for the Programmable Function is (M,N\$) and for the defining directive is (A,B\$), the value sent from the Programmable Function as M will be received by the defining directive in A and the value sent as N\$ will be received in B\$).
- All variables in the defined function are shared with the program, not just those in the variable list, and will be affected by the current value of these common variables in the program.
- Variables used in the variable list will change whenever the function is used. Therefore, variables should be reserved for use only in the defining directives variable list to avoid unexpected alteration of other program variables.
- This directive can be used in program mode or with the EXECUTE directive only.

EXAMPLES

DEF FNA (X) = X*2/4

defines the numeric function A which operates on the variable X. If the value sent to X is 2, the function returns the value 1

DEF FNB (X,Y)=X*3 + Y

defines the numeric function B which operates on the variables X and Y. If the value sent to X is 4 and to Y is 7, the function returns the value 19

DEF FNC (X\$) = (NUM(X\$))

defines the numeric function C which operates on the variable X\$. If the value sent to X\$ is "12345", this function returns the numeric value 12345

DEF FNF\$ (X\$) = X\$ + "PRTP"

defines the string function F\$ which operates on the string X\$ and the constant "PRTP". If the value sent to X\$ was "13345", this function would return the value "13345PRTP"

FUNCTION

This directive defines a numeric array of 1, 2, or 3 dimensions by establishing a unique subscripted variable name for each element of the array.

SYNTAX

DIM variable-name (1st-dimension
[,2nd-dimension][,3rd-dimension])

where:

variable-name is a single letter numeric variable name

nth-dimension is an integer value or variable specifying the largest subscript for the nth dimension of the array

DESCRIPTION

- An array must be dimensioned in a program before any reference is made to its elements.
- Dimensioned variables have only a single letter name.
- Dimensioned subscripts range from 0 to the largest value specified for the subscript.
- Each element of the array is initialized to 0 when the array is dimensioned.
- Memory space used to store an array may be released by redimensioning the array to 0.
- More than one variable may be dimensioned in a statement by separating the variable names with commas.

EXAMPLES

DIM A (X)

if $X = 5$, defines an array with the elements: A
(0) A (1) A (2) A (3) A (4) A (5)

DIM B (2,3)

defines a two-dimensional array with the
elements:

B (0,0) B (0,1) B (0,2) B (0,3)

B (1,0) B (1,1) B (1,2) B (1,3)

B (2,0) B (2,1) B (2,2) B (2,3)

DIM B (0)

releases memory space set up in example 2

DIM A (X), B (5)

if $X = 5$, sets up the same array as the first
example and an identical one of the variable
B

FUNCTION

This directive defines a string of the specified length and fills it with blanks, or, optionally, with a single character.

SYNTAX

DIM variable-name (length [,string-value])

where:

variable-name is a single letter string variable name

length is an integer value or variable specifying the number of characters assigned for the string length

string-value is a string value whose first character is used as the character that fills the string

DESCRIPTION

- If an optional string value is specified, the first character of this string is used to fill the dimensioned string.
- Memory space used to store a string may be released by redimensioning it to 0 or assigning it the null string value "".
- More than one variable may be dimensioned in a statement if the variables are separated by commas.

EXAMPLES

DIM A\$ (X)

if X=5, defines A\$ as 5 blank space characters

DIM B\$ (5,“*”)

defines B\$ as “*****”

DIM B\$ (Z,T\$)

if Z=5 and T\$="PKO", defines B\$ as
"PPPPP"

DIM B\$ (0)

releases the memory space set up for B\$ in
example 3. B\$="" would have the same
effect

DIM A\$ (5), B\$ (5)

has the same effect as the first example and
creates an identical string for the variable B\$

FUNCTION

This function returns an integer value that depends upon the value of the ERR variable that was set by the last error condition.

SYNTAX

ERR (error-code-list)

where:

error-code-list is a list of error code numbers that are used to establish the value returned by the function

DESCRIPTION

- A list of error codes is specified and the position of the code in the list determines the value returned by the function.
- The value returned is determined as follows:

If the value of the ERR variable matches the first error code on the list, the function returns 1.

If the value matches the nth error code, the function returns the value n.

If no match between the ERR variable and the error codes listed occurs, the function returns the value 0.

EXAMPLES

ERR (0,12,34)

returns the value 1 if Error 0 is produced

returns the value 2 if Error 12 is produced

returns the value 3 if Error 34 is produced

returns the value 0 if any other error is produced

Y = ERR (33,64)

if the value of the ERR variable is 64, assigns Y the value 2

Error Variable

ERR

FUNCTION

This variable is set to the number of the Error condition that last occurred during program processing.

SYNTAX

ERR

DESCRIPTION

- This variable is reset when a new error condition occurs.
- This variable is set to 0 when an error-sensitive command (i.e., READ, WRITE, etc.) is successfully executed or when a BEGIN, CLEAR, END, LOAD, RESET, RUN program-name, START, or STOP directive is processed.

EXAMPLES

E = ERR

if E = 19, the last error to occur was Error 19, which is related to Program Format or Size

FUNCTION This function conducts a bit-by-bit exclusive OR on the binary form of two string values and expresses the resulting binary value as the ASCII character that value represents.

SYNTAX XOR (string-value,string-value)

DESCRIPTION

- An exclusive OR operation is a bit-by-bit comparison of corresponding bits in both strings. The output is a 1 if either, but not both, bit being compared is a 1, and 0 if the bits being compared are both 0 or both 1.
- Both arguments must be the same length or an error is produced.
- This function is commutative.

EXAMPLES

XOR ("A","B")
returns the character "ETX" (00000011), which represents the result of the exclusive logical OR of "A" (01000001) and "B" (01000010)

X\$ = XOR (X\$,Y\$)
if X\$ = "B" and Y\$ = "A" returns the same result as the first example and assigns it to the X\$

XOR ("C",\$01\$)
returns the character "B" (01000010), which is the result of the exclusive OR of "C" (01000011) and \$01\$ (00000001)

FUNCTION This function converts a numeric value to its floating point exponential form and returns the numeric value of the exponent.

SYNTAX EPT (numeric-value)

DESCRIPTION • The setting of the PRECISION directive is disregarded when the value of this function is calculated.

EXAMPLES

EPT (1)
returns the value 1 ($1 = .1E1$)

EPT (.0234)
returns the value - 1 ($.0234 = .234E - 1$)

EPT (X)
if $X = .003024$, returns the value - 2 ($.003024 = .3024E - 2$)

EPT (- 231)
returns the value 3 ($- .231E3$)

$Z = \text{EPT}(Y/4)$
if $Y = 100$, returns the value 2 ($100/4 = 25 = .25E2$) and assigns it to the variable Z

FUNCTION This function returns the file name and additional information on a file or device that is open on the specified channel.

SYNTAX FID (channel)

DESCRIPTION • A 22-byte string is returned in the following format:

Bytes	1-3	Starting Sector (hexadecimal)
	4-9	File Name (characters 1 through 6)
	10	File Type (coded in hexadecimal) \$00\$—Indexed File \$02\$—Direct or Sort File \$04\$—Program File
	11	Key Size + Pointer Size (hexadecimal) Pointer Size = 4 for less than 32,768 records Pointer Size = 6 for more than 32,768 records
	12-14	Number of Records (hexadecimal)
	15-16	Size of Records (decimal bytes)
	17-19	Ending Sector + 1 (hexadecimal)
	20	Logical Disk Number
	21-22	File Name (characters 7 and 8)

- If a terminal or printer device is specified, only two bytes are returned. These contain the device name (i.e., LP, P1, T3, etc.).

EXAMPLES

X\$ = FID (1)

PRINT X\$ (4,6)

if the terminal prints "IND456", "IND456" is
the file open on channel 1

PRINT DEC (X\$(1,3))

if the terminal prints 241, the starting sector
address for the file is 241

FUNCTION This directive maintains 14 place computational accuracy by processing numeric values with more than 14 digits in their Floating Point exponential form and setting PRECISION to 14 significant digits.

SYNTAX FLOATING POINT

DESCRIPTION

- Values that exceed 14 significant digits are rounded.
- FLOATING POINT is cancelled by the BEGIN, CLEAR, END, LOAD, PRECISION, RESET, RUN "program-name" or STOP directives.
- FLOATING POINT or PRECISION directives must be used to process or output values with more than 2 significant digits.

EXAMPLES FLOATING POINT
processes or outputs:

.000000000087654 as .87654E - 10

1347856888333300 as .13478568883333E + 16

.6543 as .6543

FUNCTION This function returns the decimal fractional portion of a numeric value.

SYNTAX FPT (numeric-value)

DESCRIPTION • The value returned by this function is affected by the setting of the PRECISION directive.

EXAMPLES FPT (13.85)
returns the value .85

FPT (- 3.4)
returns the value - .4

Z = FPT (S)
if S = - 5.23, assigns Z the value - .23

FPT (X)
if X = .12E - 40, returns the value .12E - 40

X = FPT (T)
if T = 2.456 and:

if PRECISION is 2, assigns X the value .46

if PRECISION is 3, assigns X the value .456

FUNCTION This function converts each byte of the binary form of a string value to odd parity. The same value that is entered is returned, in its odd parity form.

SYNTAX GAP (string-value)

DESCRIPTION

- To generate odd parity, each byte is evaluated and a 1 or 0 is placed in the left-most bit to maintain an odd number of 1's in the byte.
- Odd parity is used as a means of checking for errors in data transmission.

EXAMPLES

GAP ("A")
returns "A" (11000001), the odd parity form of "A" (01000001)

X\$ = GAP ("ABC")
returns "ABC" (11000001 11000010 01000011), the odd parity form of "ABC" (01000001 01000010 01000011)

FUNCTION

This function conducts a complex logical operation on the binary form of a string value and returns a 2-byte binary string that is expressed as the characters the value represents.

SYNTAX

HSH (string-value [,2-byte-string])

where:

2-byte-string is a 2-byte string that is the result of a previous HSH operation

DESCRIPTION

- The logical operation of this function is a complex logical OR, in a predetermined pattern, of each bit of the binary form of the string.
- This function is associative, allowing results to be accumulated for use in later HSH functions. For this capability, an optional 2-byte string, which is the result of a previous HSH operation, may be specified.
- The function evaluates the argument string from left to right.
- This function is not commutative.

EXAMPLES

HSH ("A")
returns the string "00000000 01000001"

X\$ = HSH (C\$)
if C\$ = "A", has the same effect as the first example and assigns the value produced to X\$

FUNCTION This function converts a string into a string containing the hexadecimal characters of its equivalent hexadecimal code.

SYNTAX HTA (string-value)

DESCRIPTION

- Two hexadecimal characters are used to represent an ASCII character; therefore, the resultant string is twice as long as the input string.
- This function does not produce a hexadecimal string, but rather a string containing hexadecimal code characters.
- The inverse of this function is the ATH function.

EXAMPLES HTA ("A")
returns the string "41"

HTA ("ABC")
returns the string "414243"

HTA (P\$)
if P\$ = "ABC", has the same effect as the second example

FUNCTION This function returns the number of the highest disk sector available to a User Task on the specified disk.

SYNTAX HSA (disk-number)

where:

disk-number is an integer value (0-9) that specifies a logical disk

DESCRIPTION • The number of the disk sector is expressed as a numeric decimal value.

EXAMPLES H = HSA (3)
if H = 31499, the highest disk sector that is available on the logical disk 3 is sector 31499

FUNCTION This function conducts a logical inclusive OR operation on the binary form of two string values and returns the resulting binary value as the character it represents.

SYNTAX IOR (string-value,string-value)

- DESCRIPTION**
- The inclusive OR operation is a bit-by-bit comparison of corresponding bits on both strings. If either or both bits being compared are 1, the resultant bit is a 1. If both bits are 0, the resultant bit is 0.
 - Both values must be the same length.
 - This function is commutative.

EXAMPLES IOR ("A","B")
returns the character "C" (01000011) which represents the result of the logical inclusive OR of "A" (01000001) and "B" (01000010)

IOR ("B",D\$)
if D\$="A", returns the same result as the first example

X\$ = IOR ("A",\$60\$)
returns the character "a" (01100001) which represents the result of the logical inclusive OR of "A" and \$60\$ (01100000) and assigns it to X\$

FUNCTION

This function returns the index number of the record that is currently indicated by the file record pointer to the next index number or record.

SYNTAX

IND (channel [,ERR = stmt-no])

where:

channel is an integer value from 0-9 that specifies the channel number of an open file

stmt-no is the number of the statement to branch to if an error occurs while processing this function

DESCRIPTION

- Reference to this function does not update the record pointer to the next index number or record.
- Reference to the IND function produces an error if the pointer is positioned at the end of the file.

EXAMPLES

X = IND (1)

if X is assigned the value 45, then the record pointer indicates that the current record in the file to be accessed on channel 1 is the record with Index number 45

Y = IND (1,ERR = 7860)

has the same effect as the first example and, in addition, branches to statement 7860 if an error develops during processing

FUNCTION This function returns the numeric integer portion of a numeric value.

SYNTAX INT (numeric-value)

EXAMPLES INT (34.23)
returns the value 34

INT (X)
if $X = 3.56$, returns the value 3

INT (- 65.3)
returns the value - 65

INT (S/3)
if $S = 100$, returns the value 33

INT (X)
if $X = .234E + 45$, returns the value .234E + 45

$Y = \text{INT} (U/2)$
if $U = 25$, assigns Y the value 12

KEY Function

KEY

FUNCTION

This function contains the string value that is the Key value of the record pointer, indicating the current record to be accessed in the indicated file open on the stipulated channel.

SYNTAX

KEY (channel [,ERR = stmt-no] [,END = stmt-no])

where:

channel is an integer value from 0-9 specifying the channel number of an open file

stmt-no (ERR) is the number of the statement to branch to if an error occurs while processing this function

(END) is the number of the statement to branch to if the end of the file is reached

DESCRIPTION

- Reference to the KEY function will not update the record pointer to the next Key value.
- Reference to the KEY function will produce an error if the pointer is positioned at the end of the file.

EXAMPLES

X = KEY (1)

X is assigned the value "ABC"; this is the Key value of the file record pointer (the current record pointer that is the current record to be accessed for the file open on channel 1)

PRINT KEY (1,ERR = 7999)

returns the Key value for the current record and branches to statement 7999 if an error occurs while processing this function

FUNCTION

This function returns a numeric value that is equal to the number of characters in the specified string value.

SYNTAX

LEN (string-value)

EXAMPLES

LEN ("COMPUTER")
returns the value 8

LEN (A\$)

if A\$ = "COMPUTER", has the same result as the first example

X = LEN (B\$)

if B\$ = "TT000", returns the value 5

FUNCTION This directive is used to assign a value to a string or numeric variable.

SYNTAX [LET] variable-name = value

where:

LET is optional syntax that appears in the program listing

variable-name is a variable name (string or numeric)

value is a numeric or string value (constant, variable, function, or expression) that is assigned to the named variable

- DESCRIPTION**
- Multiple assignments can be made if separated by a comma.
 - This directive can be used in an IF/THEN/ELSE directive to make conditional assignments.

EXAMPLES

LET A = 6
assigns the value 6 to the numeric variable A

A = 6
has the same effect as the first example

B\$ = "TEMP"
assigns the value "TEMP" to the string variable B\$

B\$ = C\$
if C\$ = "TEMP", has the same effect as the preceding example

LET C\$ = "FILE" + Q\$

if Q\$ = "ORDERS", assigns C\$ "FILE
ORDERS"

LET B = 6 + T, G = T/20

if T = 10, assigns 16 to B and .5 to G

FUNCTION

This function converts a string that is the compiled form of a BASIC statement into its English text form.

SYNTAX

LST (string-value)

where:

string-value is the compiled form of a single BASIC statement

DESCRIPTION

- The compiled form of the statement is a shorthand representation that (a) is equivalent to the original statement and is used to conserve space when the statement is stored in memory and (b) increases processing speed when executed. Compiling occurs automatically when a statement is entered and the Carriage Return key is depressed. Compiled statements may also be produced with the CPL function.
- Conversion of a compiled statement into its English text format occurs when the statement is listed by the LIST directive, output at the terminal when an error occurs, or converted by this function.
- The inverse of this function is the CPL function.
- This function can also operate on the output of the PGM function.

EXAMPLES

LST A\$

if A\$ = \$006449C37BEB03F2\$, this function returns the string "0100 LET X = 3"

$Y\$ = \text{LST}(B\$ + C\$)$

if $B\$ = \$0064\$$ and $C\$ = \$49C37BEB03F2\$$,
assigns $Y\$$ the value "0100 LET $X = 3$ "

$\text{LST}(\text{CPL}(D\$))$

if $D\$ = "100X = 3"$, has the same effect as the
first example since both statements are
equivalent.

FUNCTION This function returns the binary logical AND of two strings.

SYNTAX AND (string-value,string-value)

DESCRIPTION

- A logical AND operation is a bit-by-bit comparison of corresponding bits in both strings. If both bits are 1, the output is 1. In all other cases, the output bit is zero.
- Both string-values must be the same length or an error is produced.
- This function is commutative.

EXAMPLES

AND ("A","B")
if "A" is binary 01000001 and "B" is binary 01000010, the result is 01000000, the character "@"

D\$ = AND (X\$,Y\$)
if X\$ = "B" and Y\$ = "A", returns the same result as the first example and assigns it to D\$

AND ("Q",\$4F\$)
if "Q" is binary 01010001 and \$4F\$ is 01001111, the result is 01000001, the character "A"

FUNCTION This function conducts a bit-by-bit exclusive OR of each byte in the binary form of the given string, and returns a binary value that is expressed as the character it represents.

SYNTAX LRC (string-value)

- DESCRIPTION**
- An exclusive OR operation is a bit-by-bit comparison of corresponding bits in both bytes. The resulting bit is a 1 if one, but not both, of the bits compared is 1. If both bits are 0 or both are 1, then the resultant bit is a 0.
 - If an argument has a single byte, it is compared with the null string "00000000".
 - This function is commutative.
 - The result of this function is used to check for errors in data transmission.

EXAMPLES

LRC ("A")

returns the character "A" (01000001) which is the result of the exclusive OR of "A" (01000001) and the null string (00000000)

LRC ("AA")

returns the character "nul" (00000000) which is the result of the exclusive OR of "A" (01000001) with itself

X\$ = LRC ("AAA")

returns the same result as the first example and assigns the value to X\$

X\$ = LRC (B\$ + C\$)

if B\$ = "A" and C\$ = "AA", has the same effect as the third example

FUNCTION This function returns the numeric remainder of a division operation carried out on the specified dividend and divisor.

SYNTAX MOD (numeric-dividend, numeric-divisor)

where:

numeric-dividend is a numeric value that is to be divided

numeric-divisor is a numeric value that is the divisor

EXAMPLES MOD (8,3)
divides 8 by 3 and returns the value 2, the remainder

MOD (X,Y)
if X = 25 and Y = 4, divides 25 by 4 and returns the value 1, the remainder

A = MOD (12,5)
assigns A the value 2, the remainder of $12 \div 5$

FUNCTION

This function returns a bit-by-bit logical inversion or NOT of the binary form of the string value. The binary value returned is expressed as the character that value represents.

SYNTAX

NOT (string-value)

DESCRIPTION

- The logical inversion or NOT operation outputs a 1 if the evaluated bit is 0 and outputs a 0 if the input bit is a 1.
- This function's operation is often referred to as a ones-complement operation.

EXAMPLES

NOT ("A")

returns the inversion of 01000001 which is 10111110, the code for the character ">"

X\$ = NOT (B\$)

if B\$ = "V", returns the value ")" and assigns it to X\$

FUNCTION This function converts a string value that contains numeric characters into its numeric form.

SYNTAX NUM (string-value [,ERR = stmt-no])

where:

string-value is a string value that contains characters representing a legal numeric value

stmt-no is the number of the statement to branch to if the function produces an error

DESCRIPTION

- Legal numeric characters include the digits 0 to 9, +, -, decimal point and E (for floating exponential forms).
- The inverse of this function is the STR function.

EXAMPLES NUM ("1234")
returns the numeric value 1234

NUM (A\$)
if A\$ = "12.34", returns the numeric value 12.34

X = NUM (C\$)
if C\$ = ".24E + 5", X is assigned the value 24000

X = NUM (B\$,ERR = 8000)
if B\$ = "\$12.34", an error is produced (\$ is not a legal numeric character) and statement 8000 is executed

NUM ("12 34")
returns the value 1234

FUNCTION

This function stores numbers in a packed format by compacting and converting a numeric variable into a string variable.

SYNTAX

PCK (numeric-value,string-length)

where:

numeric-value is a numeric literal, variable or expression with a maximum of 12 digits, including the sign.

string-length represents the length of the packed string. Enter a value from 1 to 6. The minimum length is equal to $\text{INT}((n + 1)/2)$, where n is the number of digits to be compacted.

DESCRIPTION

- If the string length specified is too small, digits will be truncated on the left.
- This function is the opposite of the UPK function.

EXAMPLES

$X = 12345678$

$X\$ = \text{PCK}(X,4)$

the numeric value "12345678" will be packed into a 4-character string

FUNCTION This function is used to scan a reference string for the occurrence of a specified substring relationship and return a numeric value that indicates its position.

SYNTAX POS (search-string relational-operator
reference-string [,step-value] [,occurrence])

where:

search-string a string value that is the substring to be searched for in the reference string

relational-operator a valid relational operator, i.e., < , <= , = < , > , >= , = > , = , < > , > <

reference-string a string value that is the string to be searched

step-value a numeric value that specifies the size of the steps (the number of characters to skip after each test) that are used to search the reference string. Defaults to 1 if not specified

occurrence a numeric value that specifies the occurrence of the relation whose position value is returned. If not specified, this value defaults to the 1st occurrence

DESCRIPTION

- The positional value returned is dependent upon the relational operator specified and indicates the position of the first character of the substring that satisfies the indicated relation.
- Relationships are established on the basis of ASCII code sequence. Consult an ASCII table for additional information on the values assigned to characters.
- If the relational condition is not found, the value returned is 0.

EXAMPLES

For the string A\$ = "XXSUBSTRXX"

POS ("S" = A\$)
returns the value 3

POS ("S" = A\$,1,2)
returns the value 6, the position of the second "S"

POS ("S" > A\$)
returns the value 5 (the position of B, the first character in A\$ that satisfies the relation of being less than "S")

POS ("Z" < A\$)
returns the value 0 (no substring of A\$ satisfied the relation of being greater than "Z")

X = POS ("SB" > = A\$)
assigns X the value 5 (the position of "B", the first character in the substring that satisfied the relation of being less than or equal to "SB")

FUNCTION This directive sets the number of significant digits to the right of the decimal point used for processing or outputting numeric values.

SYNTAX PRECISION x

where:

x is an integer value from 0 to 14 that sets the number of significant digits to the right of the decimal point used in handling numeric values

- DESCRIPTION**
- Values with more than the allowed number of significant digits are rounded accordingly.
 - This directive and rounding come into effect only after a numerical calculation or when a value is to be output.
 - Values that are assigned by a constant (i.e., $X = .45673$) are not rounded when stored if they exceed the precision setting and are used in all processing (except printing) with the value as defined, never rounded.
 - If PRECISION is not specified, the default value is set at 2.
 - PRECISION is reset to the default value of 2 when a BEGIN, CLEAR, END, LOAD, RESET, RUN "program-name", or STOP directive is executed.
 - PRECISION is set at 14 when the FLOATING POINT directive is active.

EXAMPLES

PRECISION 4

under this setting:

if $X = 2.112233 * 2$, X is stored or output as
4.2245

if $Y = 3.445566$, Y is stored as 3.445566
Y is output as 3.4456

PRECISION X

if $X = 12$, 12 significant digits are allowed in
processing

FUNCTION This variable returns a string value that contains the name of the program currently in, or being processed by, the User Task.

SYNTAX PGN

EXAMPLES P\$ = PGN
if the name of the program currently being processed is "INDEX", P\$ is set to equal "INDEX"

FUNCTION Returns a numeric value that is the size of the program currently in or being processed by the User Task. The value returned is expressed as the decimal number of bytes.

SYNTAX PSZ

EXAMPLES X = PSZ
if X is assigned the value 3456, the size of the current program is 3456 bytes

FUNCTION This function returns the compiled form of the specified BASIC statement from the current program.

SYNTAX PGM (stmt-no)

where:

stmt-no is an integer value specifying a statement number

- DESCRIPTION**
- Compiling occurs automatically when a statement is entered and the Carriage Return key is depressed.
 - Compiled statements are returned to their English text format when the statement is listed by the LIST directive, output at the terminal when an error occurs, or converted by the LST function.
 - If the specified statement number does not exist, the next highest statement is returned.
 - If a statement number past the last statement is specified, the statement END is returned.
 - The compiled form of a BASIC statement may contain unprintable characters.

EXAMPLES

PGM (100)

if statement 100 is $X = 3$, a string whose hexadecimal form is \$006449C37BEB03F2\$ is returned

PGM (X)

if $X = 100$, has the same effect as the first example

$Y\$ = \text{PGM}(X)$

if $X = 100$, assigns $Y\$$ the same value as the second example

FUNCTION This function is used to send values to a user-programmed function that has already been defined by the DEF FN directive.

SYNTAX FN x (value-list)
FN x\$ (value-list)

where:

x/x\$ is a single character from A to Z that specifies the name of a user defined function

value-list is a list of values sent to the defining directive's variable list

- DESCRIPTION**
- Any number of numeric and/or string values may be sent, depending upon the variable list specified in the DEF FN directive, but only one value (numeric or string) is returned.
 - The values in the value list of this function and in the variable list of the defining directive may have different variable names, but must be in corresponding positions in both lists. The first variable in the Programmable Function value list sends its value to the first variable in the defining directive's variable list, etc. (i.e., if the value list for the Programmable Function is (M,N\$), and for the defining directive is (A,B\$), then the value sent from the Programmable Function as M will be received by the defining directive in A, and the value sent as N\$ will be received in B\$).

EXAMPLES

$Y = \text{FNA}(X)$

if $X = 3$ and $\text{DEF FNA}(Z) = Z * 2 / .5$, assigns Y the value 12

$\text{FNC\$}(G\$)$

if $G\$ = \text{"A"}$ and $\text{DEF FNC\$}(X\$) = \text{"FILE:"} + X\$$, returns the value "FILE: A"

$\text{FNP}(X\$,Y)$

if $X\$ = \text{"A"}$, $Y = 32$ and $\text{DEF FNP}(P\$,Q) = \text{DEC}(P\$) + Q$, returns the value 97

$\text{FNM}(N,O,P,Q)$

if $N = 2$, $O = 4$, $P = 2$, $Q = 10$, and $\text{DEF FNM}(A,B,C,D) = (A + B) / (C + D)$, returns the value .5

$\text{FNA}(1,2+3,X)$

if $X = 0$ and $\text{DEF FNA}(B,C,D) = B + C + D$, returns the value 6

FUNCTION This function provides a listing of Public Programs in the specified memory bank.

SYNTAX PUB (bank-number)

where:

bank-number is an integer value specifying a memory bank

DESCRIPTION

- The function returns a 12-byte string with the following information for each program:

Bytes 1-2 Starting memory location within the memory bank, expressed as a hexadecimal address

3-4 Program size in hexadecimal bytes

5-12 Program Name

- All configured memory banks may be accessed.

EXAMPLES

X\$ = PUB (1)

PRINTX\$ (5,5)

if the program prints "INDEX", then "INDEX" is the Public Program in memory bank 1

PRINT DEC (X\$(3,2))

if the program prints 548 then the program is 548 bytes in length

FUNCTION This function returns the size of sectors on the specified logical disk. The value returned is the number of bytes expressed as a numeric decimal value.

SYNTAX SSZ (disk-number)

where:

disk-number is an integer value, from 0 to 9, that specifies a logical disk

EXAMPLES S = SSZ (3)
if S = 1024, disk number 3 has a sector size of 1024 bytes

FUNCTION This directive assigns a value to the variable DAY that indicates the current or desired date.

SYNTAX SETDAY (string-value)

where:

string-value is an eight-byte string value
sentencing a date (intended form is
mm/dd/yy)

- DESCRIPTION**
- This variable should (on some systems, must) be set in the format mm/dd/yy.
 - On some systems there is a check for a valid date, while on others any valid string can be entered.
 - Terminating a task with the RELEASE directive resets the value of the Task variable value of DAY to the System variable value for DAY.
 - This directive, issued by any Task, affects the system wide value of DAY.

EXAMPLES SETDAY "04/26/82"
assigns the Task variable DAY, the value
representing April 26, 1982

SETDAY D\$
if D\$ = "04/26/82", has the same effect as the
first example

FUNCTION This directive assigns a numeric value to the variable TIM indicating the current or desired time.

SYNTAX SETTIME (numeric-value)

where:

numeric-value is a numeric value indicating a decimal, 24-hour based time

- DESCRIPTION**
- The value for the time is set on a 24-hour day schedule and in the decimal PRECISION 4 form of HH.hhhh [i.e., times run from 00.0000 (midnight) to 23.9999 (11.9999 pm)].
 - On some systems, valid values are between 0 and 23.9999. On other systems, any value may be entered and multiples of 24 are added or subtracted to obtain an appropriate value between 0 and 24.
 - The value of the TIM variable is updated by the system.
 - Terminating a Task with the RELEASE directive will reset the Task variable value for TIM to the System variable value for TIM.
 - This directive, issued by any Task, affects the system wide value of TIM.

EXAMPLES

SETTIME 8.5

assigns the variable TIM the value 8.5000, representing 8:30 A.M.

SETTIME X

if X = 20.5, TIM is set to indicate 8:30 P.M.

FUNCTION

This function returns a numeric value that indicates the sign of a numeric value. The values returned are:

- 1 if the value is negative
- 0 if the value is 0
- 1 if the value is positive

SYNTAX

SGN (numeric-value)

EXAMPLES

SGN (23)

returns the value 1 for positive

SGN (-.34)

returns the value - 1 for negative

SGN (X*4)

if X = - 3, returns the value - 1 for negative

Z = SGN (INT(.76))

assigns Z the value 0

FUNCTION

This function converts a numeric value into a string value with the same sequence of numeric characters. The digits of the numeric value form the characters of the new string value.

SYNTAX

STR (numeric-value [:format-mask]
[,ERR = stmt-no])

where:

format-mask is a string value that serves as the mask to be used in forming the result of this function

stmt-no is the number of the statement to branch to if the function produces an error condition

DESCRIPTION

- If a mask is specified that is too small for the numeric value, it will be ignored.
- The inverse of this function is the NUM function.

EXAMPLES

STR (654)
returns the string "654"

STR (X)
if X = 34.56, returns the string "34.56"

X\$ = STR (A)
if A = .34E + 34, X\$ is assigned the value
".34E + 34"

STR (X: "\$#,###.##")
if X = 1234.56, returns the string "\$1,234.56"

System Serial Number Variable

SSN

FUNCTION This variable returns an integer numeric containing the serial number of the SMC BASIC copy under which the system is currently operating.

SYNTAX SSN

DESCRIPTION • This variable returns the value 0 if the media are not serial-numbered.

EXAMPLES PRINT SSN
 601485

FUNCTION This variable returns the release number of the copy of BASIC under which the system is currently operating.

SYNTAX SYS

DESCRIPTION • A 6-byte string is returned in the following format:

Bytes 1 is the manufacturer's code (i.e.,
"M" for MICRO-FIVE)
2-6 is the Release number

- If the release number does not occupy 5 characters, this variable is right filled with spaces.

EXAMPLES PRINT SYS
M6.0

PRINT HTA (SYS)
4D352E313420

FUNCTION This function returns information on the status of certain values that are generated dynamically during processing of a Task. These values relate to error and escape processing and are continuously updated by the appropriate directives and system processes.

SYNTAX TCB (n)

where:

n is an integer from 0 to 9

DESCRIPTION • The function returns these values:

TCB (n) n = 0,1,2,8,9 not used and returns 0

TCB (3) returns the last system error code

TCB (4) returns the current statement number

TCB (5) returns the number of the statement to RETRY

TCB (6) returns the statement that SETESC is set to

TCB (7) returns the statement that SETERR is set to

EXAMPLES

X = TCB (6)

if X is assigned the value 5678, the statement that the SETESC directive is set to is 5678

FUNCTION This function returns the memory parameters of the active Tasks that are located within the specified memory bank.

SYNTAX TSK (bank-number)

where:

bank-number is an integer value specifying a memory bank

DESCRIPTION • This function returns a 6-byte string for each User Task in the following form:

Bytes 1-2 Starting memory location within the memory bank, expressed as a hexadecimal address

3-4 The amount of memory allocated to a task expressed as the hexadecimal number of bytes

5-6 Task ID

- All configured memory banks may be accessed.

EXAMPLES

X\$ = TSK (1)

PRINTX\$ (5,2)

if the terminal prints "T0", then T0 is the only active Task in memory bank 1

PRINTDEC (X\$(3,2))

if the terminal prints 1024, then 1024 bytes of memory have been allocated to the Task

FUNCTION This function returns a listing of all Tasks and Terminal devices configured on the system, along with their current status.

SYNTAX TSK (0)

DESCRIPTION • This function returns a 6-byte string in the following form for each task or device:

Bytes 1-2	Task or Device ID
3	Status of the Task or Device, coded: 0—Inactive 2—Active
4-6	Unused, returns \$000000\$ (not printed)

EXAMPLES T\$ = TSK (0)
if T\$ = "T00T12T32LP2", then the mask or device named T0 is inactive and T1, T3, and LP are active

FUNCTION This variable returns information on the status of certain values that are related to error and escape processing within the current Task.

SYNTAX TSM

DESCRIPTION

- This variable is set only after a Carriage Return is used to respond to a "PROGRAM EXCEPTION ENCOUNTERED" message or an "E" and 'CR' is used to respond to an ESCAPE REQUESTED message. It may then be used to send parameters to an error-processing program.

- This variable provides 32 bytes of information in the following format:

Bytes	1	Status Code
		0 = ERROR
		1 = ESCAPE
	2-9	Program Name (padded with nulls)
	10-17	Not used
	18-22	Number of statement being processed at error or escape
	23-27	BASIC error code
	28-32	System error code

EXAMPLES

X\$ = TSM

PRINTX\$ (1,17)

if the program prints the value "1INDEX", the current program being processed ("INDEX") was terminated with an Escape

FUNCTION

This variable returns the time currently being used as the System Time.

SYNTAX

TIM

DESCRIPTION

- The output is a numeric value that may be expressed up to PRECISION 4 as HH.hhhh.
- The time is based upon a 24 hour system (0.0 to 23.9999) with 0 representing midnight and 12.00 representing noon.
- The time variable is maintained by the operating system and is updated approximately every 1/10,000 of an hour.
- The accuracy of the time will be affected by the setting of the PRECISION directive.
- This variable affects the system wide value of TIM when issued by any Task.

EXAMPLES

T = TIM

at 11:30 A.M., T is assigned the value 11.5000
at 11:45 P.M., T is assigned the value 23.7500

FUNCTION This function unpacks a previously packed numeric variable.

SYNTAX UPK (string-value)

where:

string-value is a character variable containing a previously packed numeric value.

DESCRIPTION • This function is the opposite of the PCK Function.

EXAMPLES X = UPK (X\$)
where X\$ is the output from a previous PCK function, has a length of 4. X is assigned the original numeric variable 2345678 (a length of 8).



CHAPTER THREE—SYSTEM DIRECTIVES

The interface between a user or a program and the computer and its operating system is controlled by a group of directives known as System Directives. These directives manage the start-up, execution, termination, and the environment of these processes. Thoroughbred/OS supports two types of processes: Task and Programs. This chapter describes the directives used by these two processes.

TASKS

Since Thoroughbred/OS can support multiple users, there must be a means of establishing each user-program-computer interaction as unique, so that confusion does not occur when processes are carried out at the same time within the system. The three categories of tasks are:

- **System Control Task**

The System Control Task refers to a task that has a priority status, allowing it to direct and override some capabilities of the system and other Tasks. This is a special type of User Task.

- **User Task**

A User Task refers to a user interface with the computer that interacts through a terminal. All User Tasks, excluding the System Control Task, have equal status.

- **Ghost Task**

A Ghost Task refers to a "Background" Task. These tasks are usually programs that, once started from a User Task, may not interact with the User Task terminals.

A task may process a program or provide for an interface between the user and the computer, usually through a Video Display Terminal (VDT). Each user comprises a single User Task.

Thoroughbred/OS uses certain task designations when initiating or terminating tasks or referring to tasks. These serve as names to uniquely identify the tasks that are operating on the system. The designations are:

“T0” through “TF” —user or terminal tasks, usually limited to 16 (hexadecimal), “T0” is usually configured as the System Control Task
“G0” through “GF”—ghost or background tasks, limited to a total of 16 (hexadecimal), including user tasks
“SY” — synchronous communication task

Task Directives:

START
RELEASE

PROGRAMS

A program is a series of BASIC statements, consisting of directives and other language components, that are executed in sequence to perform a logical process. Each task can run a single program at a time.

Memory is provided to each task for program loading, processing and data storage. Program processing occurs in Program mode. In this mode, the operating system controls the computer according to program logic.

Program Mode Processing

The system directives that support a program can be classed as (a) processing directives that initiate, execute or terminate the program and affect program parameters, and (b) contents directives that operate on or alter the program contents within memory. These directives are:

Program Processing Directives

BEGIN
CLEAR
END
ESCAPE
LOAD
RESET
RUN
START
STOP

Program Contents Directives

DELETE
EDIT
ENDTRACE
EXECUTE
LIST
MERGE
REM
SETTRACE

Console Mode Processing

Most Thoroughbred/OS directives can be used in both Program Mode and in an alternate interactive mode, called Console Mode. In Console Mode, each statement is processed by the system as it is entered. Statements do not have line numbers and control is returned to the user after the processing of each directive is complete. Multiple directives can be processed in this mode by separating them with a semicolon (;). Copies of the directives entered in this way, and the order in which they were processed, are not kept.

Program Mode can be terminated, and Console Mode initiated, using the ESCAPE directive or by depressing the ESCAPE key (as long as the SETESC directive has not been set).

Escape Processing

When escape processing is initiated, the program will compute and print the interrupted statement and initiate Console Mode. The task area remains intact and processing may be resumed using the RUN directive.

BEGIN Directive

BEGIN

FUNCTION This directive initializes certain program parameters. It is the most comprehensive in a series of similar directives that includes CLEAR and RESET.

SYNTAX BEGIN

DESCRIPTION

- The exact effects of this directive are:
 1. Closes all files and devices
 2. Initializes all variable values to 0 or "" (null string)
 3. Clears the return address stack (used to hold address values for certain directives, i.e., NEXT, RETURN, RETRY, etc.) RETRY, etc.)
 4. Sets value of ERR, CTL to 0
 5. Sets PRECISION to 2, ends FLOATING POINT
 6. Sets SETERR and SETESC to 0

EXAMPLES BEGIN
affects the program parameters as detailed above

FUNCTION This directive initializes certain program parameters. It is the intermediate in a series of similar directives that include BEGIN and RESET.

SYNTAX CLEAR

DESCRIPTION

- The exact effects of this directive are:
 1. Initializes all variable values to 0 or "" (null string)
 2. Clears the return address stack (used to hold address values for certain directives, i.e., NEXT, RETURN, RETRY, etc.)
 3. Sets value of ERR, CTL to 0
 4. Sets PRECISION to 2, ends FLOATING POINT
 5. Sets SETERR and SETESC to 0

EXAMPLES CLEAR
affects the program parameters as detailed above

FUNCTION This directive removes statements from a program.

SYNTAX DELETE [1st-stmt-no] [,] [2nd-stmt-no]

where:

1st-stmt-no is the number of the first statement to be removed

2nd-stmt-no is the number of the last statement to be removed

DESCRIPTION

- Only the copy of the program in the Task memory is altered; the copy of the program stored on disk is not altered unless the program is saved.
- The program statement or a range of statements to be removed can be specified.
- The following default values are provided for unspecified statement numbers:

1st-stmt-no = default value is 1

2nd-stmt-no = default value is 9999

The optional comma specifies the use of the default values for unspecified values.

- If statement numbers are not specified, all statements are removed, effectively erasing the program from Task memory.

EXAMPLES

DELETE

removes all program statements from the Task memory area, i.e., same as DELETE 1,9999 or DELETE 1, or DELETE ,9999

DELETE 123

removes program statement 123

DELETE 123,

removes all statements from 123 to 9999

DELETE 123,350

removes all program statements from 123 to
350

DELETE ,350

removes all program statements from 1 to
350

FUNCTION

This directive provides a means of editing program statements without retyping the entire statement and provides for copying, replacing, deleting, and inserting strings of text in the statement.

SYNTAX

EDIT stmt-no edit-specifier [string-constant] . . .

where:

stmt-no is the number of the statement to be edited

edit-specifier is a letter or blank specifying the editing procedure to conduct on the statement

string-constant is a string constant (not enclosed by " ") that specifies text to be manipulated

DESCRIPTION

- Edit commands operate on the text of a program statement at a position specified by the edit pointer. This pointer position is established by the edit operations as the next character after the last character of the string constant that is specified or manipulated.

- The edit specifiers are:

C—copies text from the edit pointer to the last character of the specified string

R—replaces text with the specified string starting at the edit pointer

D—deletes text from the edit pointer to the last character of the specified string

—(absence of an edit specifier) inserts the specified string (in brackets) as text starting at the edit pointer

'CR' (carriage return) instructs the system to copy the rest of the statement starting at the edit pointer

- Spaces must be matched in the C and D edit specifiers.
- Spaces inserted by the D and insert specifiers will be included, depending upon statement syntax requirements.
- This directive can be used only in Console Mode or with the EXECUTE directive.

EXAMPLES

```
*ERR          V
00100 IF X = 10 ORRY > = 123 THEN GOTO 678
```

EDIT 100 C[OR]D[R]

produces 00100 IF X = 10 OR Y > = 123 THEN GOTO 678 (the statement is copied to OR, the edit pointer is positioned after the R, the string from the edit pointer to the occurrence of R is deleted, the Carriage Return copies the remainder of the statement)

EDIT 100 C[OR] [X < 5 OR]C[6]R[99]

on the result of example 1 produces 00100 IF X = 10 OR X < 5 OR Y > = 123 THEN GOTO 699 (the string is copied to OR with the edit pointer positioned after the R, the specified string is inserted, the remainder of the string is copied to the 6, the edit pointer is placed after the 6, and the specified string replaces the next two characters in the statement)

EDIT100C[OR]C[OR]D[Y] [X1]

on the result of example 2 produces 00100 IF
X = 10 OR X < 5 OR X1 > = 123 THEN GOTO
699 (in order to get to the second OR, the first
is copied to advance the edit pointer to
where the next specification of OR will indi-
cate its second occurrence)

END Directive

END

FUNCTION

This directive terminates a program and initializes certain program parameters.

SYNTAX

END

DESCRIPTION

- This directive has the same effect as the STOP directive. In addition, it also terminates a MERGE directive.
- The effects of this directive are:
 1. Closes all files and devices
 2. Sets the execution pointer to the first statement
 3. Performs a RESET
 - Return address stack cleared
 - ERR and CTL set to 0
 - PRECISION set to 2, ends FLOATING POINT
 - SETERR and SETESC reset to 0
 4. Does not clear Task data variables
 5. Does not alter Task memory contents
 6. Ends SETTRACE

EXAMPLES

END

affects the program and Task parameters as detailed above

ENDTRACE Directive

ENDTRACE

FUNCTION This directive ends a Trace, initiated by the SETTRACE directive.

SYNTAX ENDTRACE

DESCRIPTION • A Trace initiated by the SETTRACE directive is also terminated by an END or STOP directive.

EXAMPLES ENDTRACE
 terminates the listing of statements as they are executed

FUNCTION This directive interrupts and suspends program execution.

SYNTAX ESCAPE

DESCRIPTION

- All program and Task parameters are unaffected by this directive.
- The following effects are produced when this directive is executed in a program:
 1. Returns Task Control to Console Mode
 2. Prints the statement containing the ESCAPE directive
 3. Sets the program execution pointer at the statement following the ESCAPE directive
- Program execution can be resumed from Console Mode by typing RUN. The next program statement can be displayed, without executing, by typing ESCAPE.
- Escape key has same effect as ESCAPE Directive if the program does not contain a SETESC directive. However, there is no relationship between the ESCAPE and SETESC directives.

EXAMPLES ESCAPE
terminates program execution and retains all program parameters as listed above

FUNCTION This directive executes a data string consisting of a BASIC statement.

SYNTAX EXECUTE string-value

where:

string-value is a string constant, variable or expression that forms a BASIC statement

- DESCRIPTION**
- This directive allows a valid BASIC statement to be handled as data. When this directive is executed, the "data" (a BASIC statement) is included in the program being processed. This allows the form of a program to be altered by processing statements as data during program execution, rather than by interrupting the program and respecifying program logic.
 - If the BASIC statement used as data does not include a statement number, it is treated as a Console Mode directive and is executed immediately.
 - If the BASIC statement used as data includes a statement number, the statement is inserted into the program and will appear in a listing of the program and will be saved if a SAVE directive is executed.
 - If this directive specifies a statement with a statement number that already exists in the program, it replaces the pre-existing statement.

- A directive that is only allowed in Console Mode may be executed in a program if it is entered into the program with this directive. (Note: EDIT but not LOAD or FOR can be executed in this manner.)

EXAMPLES

EXECUTE "X = 34"

this directive will execute the assignment directive X = 34 immediately

EXECUTE F\$

if F\$ = "4467 X = 54 + H", will insert the statement X = 54 + H into the program as statement 4467; it is executed when the program processes statement 4467

EXECUTE A\$ + B\$ + C\$

if A\$ = "4467", B\$ = "X =", C\$ = "54 + H", has the same effect as example 2

EXECUTE "990 GOTO" + D\$

if D\$ = "1000", the effect is to introduce statement 990 GOTO 1000 into the program

FUNCTION This directive outputs program statements from the copy of the program currently in memory to a file or device OPEN on the channel specified.

SYNTAX LIST (channel [,ERR = stmt-no] [,IND = index-no] [,TBL = stmt-no]) [1st-stmt-no] [,] [2nd-stmt-no]

where:

channel is an integer value specifying the channel of an OPEN file or device

stmt-no (ERR) is the number of the statement to branch to if an error occurs while processing this directive

stmt-no (TBL) is the number of the statement in which the TABLE statement for code conversion is located

index-no is an integer value specifying the Index number of the first record to be written to

1st-stmt-no is the first statement number to be listed

2nd-stmt-no is the last statement number to be listed

DESCRIPTION • The following default values are set if statement numbers are not specified:

1st-stmt-no = default value is 1

2nd-stmt-no = default value is 9999

The optional comma specifies the use of default values for unspecified statement numbers.

- If the output is directed to a file for later use with a MERGE directive, the file must have a specialized format (refer to MERGE).
- Listing of statements on the terminal may be ended by using the Escape Key or with the Control S sequence. Listing may be re-initiated with the Control Q sequence.

EXAMPLES

LIST

lists all program statements on the terminal, i.e., same as LIST 1,9999 or LIST 1, or LIST ,9999

LIST 123

lists statement 123

LIST 123,

lists statements 123 to 9999

LIST ,123

lists statements 1 to 123

LIST 123,350

lists statements from 123 to 350

LIST (4)123,350

has the same effect as example 5 but lists the output to the file or device OPEN on channel 4

LIST (4,ERR = 7999,TBL = 5500)123,350

has the same effect as example 6; also transfers program execution to statement 7999 if an ERROR occurs while processing this directive and refers to the TABLE statement at line 5500 for code conversion information

FUNCTION This directive transfers a copy of a program from disk storage into Task memory.

SYNTAX LOAD program-name

where:

program-name is a string value that specifies a program name

- DESCRIPTION**
- If a program is loaded without error, the following occurs:
 1. Task program memory area is purged
 2. The RESET Directive is executed
 3. A copy of the program from disk storage into the User Task Program memory area is loaded
 4. The Program Execution Pointer is positioned to the first statement
 5. The program is not executed
 6. User Task data variables are not cleared
 - If the specified program cannot be loaded, the Task area is not altered.
 - This directive can be used in Console Mode only and cannot be executed with the EXECUTE directive.

EXAMPLES

LOAD "INDEX"
loads the program named "INDEX"

LOAD P\$
if P\$ = "INDEX", has the same effect as the first example

FUNCTION

This directive allows you to combine program statements input from a terminal or a special LIST file with the program currently in the User Task area.

SYNTAX

MERGE (channel [,ERR = stmt-no]
[,IND = index-no] [TBL = stmt-no])

where:

channel is an integer value specifying the channel of an OPEN file or device

stmt-no (ERR) is the number of the statement to branch to if an error occurs while processing this directive

stmt-no (TBL) is the number of the statement in which the TABLE statement for code conversion is located

index-no is an integer value specifying the Index value of the first record to be read from

DESCRIPTION

- Program statements are combined by sequentially reading records, each record being a valid program statement from an input source. When the reading of the program statement records is ended, the records are combined into the current program, and execution of the program continues.
- The input source for the program statements may be either a terminal or a specially constructed INDEXED file (LIST file). The format of this file is:

- a. An INDEXED file with a record size of 80 bytes
 - b. One program statement per record with trailing null or space characters
 - c. Bytes 1-4 must contain a numeric statement number; byte 5 of a continuation statement must be a colon (:)
- Program statements that are merged with the current program are processed according to the following:
 1. Unnumbered program statements cannot be executed
 2. All numbered program statements are sorted in statement number order
 3. In case of a conflict between statement numbers, the last statement to be entered replaces any previous statements with the same number
 4. Program statements are retrieved sequentially until an END directive is encountered
 5. An END directive terminates the MERGE
 6. An END directive input from a terminal does not need a statement number
 - Syntax errors are not reported for merged statements.

EXAMPLES

MERGE (0)

interrupts program execution to allow input of program statements from the TASK terminal

MERGE (2)

interrupts program execution to allow input of program statements from the LIST file OPEN on channel 2

MERGE (2,ERR = 7999,IND = 20,TBL = 50)

has the same effect as the second example, but starts reading at the record with Index number 20; transfers program execution to statement 7999 if an error occurs while executing this directive; refers to the TABLE statement at statement 50 for the code conversion information

FUNCTION This directive terminates a Task's operations and de-allocates its memory.

SYNTAX RELEASE [task-id]

where:

task-id is a string value that specifies the Task ID

DESCRIPTION

- The System Control Task may specify any Task.
- All other Tasks may only specify themselves or a Ghost Task they initiated.

EXAMPLES

RELEASE
terminates the Task issuing the directive and de-allocates memory assigned to that Task

RELEASE "T3"
terminates the Task named "T3" and de-allocates memory assigned to that Task. This directive can be issued only by the System Control Task or Task "T3"

FUNCTION This directive designates a commentary statement (non-executable).

SYNTAX REM [comment]

DESCRIPTION

- A REM directive placed at statement number 10 will be printed by the Utility programs that list program descriptions.
- In a continuation statement, any directive to the right of the REM directive is considered part of the comment and will not be executed.
- This is the only directive that can appear at the end of compound statements which produce an unconditional branch without causing a syntax error.

EXAMPLES

00010 REM PROGRAM TO CALCULATE
VALUES

this statement is not executed, but serves to
identify the program

X = 45; REM SETS VALUE OF PAGE LENGTH
TO 45

this REM directive serves as a comment to
the action of the statement. A REM placed
before the assignment will cause the assign-
ment not to be executed

FUNCTION This directive initializes certain program parameters. It is the least comprehensive in a series of similar directives that include BEGIN and CLEAR.

SYNTAX RESET

DESCRIPTION

- The exact effects of this directive are:
 1. Clears the return address stack (used to hold address values for certain directives, i.e., NEXT, RETURN, RETRY, etc.)
 2. Sets value of ERR, CTL to 0
 3. Sets PRECISION to 2, ends FLOATING POINT
 4. Sets SETERR and SETESC to 0

EXAMPLES RESET
initializes the program parameters as detailed above

FUNCTION This directive initiates or resumes program execution.

SYNTAX RUN [program-name] [,ERR = stmt-no]

where:

program-name is a string value that specifies a program name

stmt-no is the number of the statement to branch to if an error occurs while processing this directive

DESCRIPTION

- If a program name is specified, an attempt to LOAD the program is made. If the program is successfully loaded, the following occurs:

1. The User Task memory area is purged
2. A RESET is performed (open files are not closed and Task data variables are not cleared)
3. Execution of the program is started

- If the program is not successfully loaded, the Task area remains unaltered.
 - If no program name is specified, the program presently in Task memory is executed, starting with the statement indicated by the execution pointer.
 - This directive is also used to resume execution following an escape or error which terminates program execution. In this case, the following takes place:
-

1. Program Execution Pointer is not moved
2. Open files and devices are not closed
3. User Task Data (variables) are not cleared
4. Return Address Stack (NEXT, RETURN, RETRY) is not cleared
5. PRECISION and Task Variables are not set
6. SETERR and SETESC are not reset

EXAMPLES

RUN "INDEX"

executes a LOAD directive for the program "INDEX" and starts execution at the first program statement

RUN A\$

if A\$ = "INDEX", has the same effect as the first example

RUN

starts execution of the program currently in the User Task area at the statement following the position of the current Program Execution Pointer

SETTRACE Directive

SETTRACE

FUNCTION This directive initiates a Trace: an output listing of program statements as they are executed.

SYNTAX SETTRACE [(channel)]

where:

channel is an integer value that specifies the channel of an OPEN file or device

DESCRIPTION • A Trace initiated by the SETTRACE directive is terminated by an ENDTRACE, END, or STOP directive.

• The output of this directive, when displayed at the terminal, may be temporarily halted using the CTL S sequence and resumed using CTL Q.

EXAMPLES

SETTRACE
initiates outputting of the listing of executed program statements to the terminal

SETTRACE (2)
initiates listing of executed program statements to the file or device designated on channel 2

FUNCTION This directive initializes and allocates memory for a Task.

SYNTAX START pages [,ERR = stmt-no] [,BNK = bank-no]
[,program-name] [,task-id]

where:

pages is an integer value from 1 to 128 that indicates the number of pages of memory to allocate

stmt-no is the number of the statement to branch to if an error occurs while processing this directive

bank-no is an integer value indicating the memory bank to assign the Task to

program-name is a string value indicating the name of a program to RUN after the Task is initialized

task-id is a string value specifying the Task ID to START

- DESCRIPTION**
- Certain features of this directive may be reserved for the System Control Task and may produce an error if specified by another Task.
 - The amount of memory specified is a minimum number of pages of 256 bytes each that is allocated for loading of a program and is not available for data. Additional memory is dynamically allocated and released as required by the operation of the User Task.

- An unspecified task-id starts the Task that issued the directive.
- Any task that has the same ID as the one specified to start is released.
- All configured memory banks may be accessed.

EXAMPLES

START 1

releases the memory assigned to the Task executing this directive and initiates a new Task with one page of memory. This is the usual size of task initiation that will avoid reserving excessive space

START 1,ERR = 8000,BNK = 2,"INDEX","T2"

has the same effect as the first example and assigns the memory to Bank 2 and starts Task "T2". In addition, branches to statement 8000 if an ERROR occurs while processing this directive; executes the program "INDEX" after the Task is initialized

STOP Directive

STOP

FUNCTION This directive terminates program execution and initializes certain Task parameters.

SYNTAX STOP

DESCRIPTION

- This directive has the same effect as the END directive except that it doesn't terminate a MERGE directive.
- The specific effects of this directive are:
 1. Closes all files and devices
 2. Sets the execution pointer to the first statement
 3. Performs a RESET
 - Return address stack cleared
 - ERR and CTL set to 0
 - PRECISION set to 2, ends FLOATING POINT
 - SETERR and SETESC reset to 0
 4. Does not clear Task data variables
 5. Does not alter Task memory contents
 6. Ends SETTRACE

EXAMPLES STOP
terminates the current program and affects the parameters as listed above

CHAPTER FOUR—PROGRAM CONTROL

Thoroughbred/OS programs are usually processed by executing each statement sequentially, according to its statement number, from statement 1 up to statement 9999. Statement numbers that do not exist are skipped and the next highest numbered statement is executed. However, certain program procedures require that the ordinary sequence of processing be altered. Thoroughbred/OS provides directives that control the order of program flow for the following:

- Conditional and unconditional branching
- Conditional execution of directives
- For/Next loops
- Subroutines
- Error branching
- Timed suspension of program processing
- Transfer of program control to Console Mode
- Compound statements
- Return address stack

PROGRAM PROCESSING

Programs are processed by executing one statement at a time, usually in sequential order. Certain program control directives may be used to effect nonsequential processing by specifying a statement to execute other than the next statement. If the statement that is specified is not present, the next highest statement after the specified statement is executed. Loops do not specify statement numbers explicitly, but implicitly they establish a starting and ending sequence of statements that are processed recursively. There are also certain directives that are used to implicitly set the number of the next statement to execute as the first statement in the program (refer to RUN, END, START and STOP).

Compound Statements

Compound statements may be constructed with multiple directives, separated by semicolons (;). BASIC statements, including compound statements, are processed from left to right. When compound statements are constructed of directives that conditionally or unconditionally transfer program control, careful consideration

must be given, since the transfer of program control may prevent execution of later portions of the statement. Examples of compound statement usage:

100 GOTO 567; X = 56

The X = 56 portion of the statement is never executed, since the GOTO transfers control to statement 567, and the assignment directive in the statement is never reached.

200 IF X = 5 THEN GOSUB 46; PRINT "END OF DATA"

If X is equal to 5, the program takes a subroutine branch to statement 46. When the subroutine returns (via RETURN), the PRINT statement is executed. If X is not equal to 5, the next higher numbered statement is executed and the PRINT statement never gets processed.

The following directives must appear only at the end of a compound statement; any directives that appear after them will not be executed:

END	GOTO	RETRY
EXIT	ON GOTO	RETURN
EXITTO	REM	STOP

NOTE: A REM statement may follow any of the above directives since it is not an executable statement.

Return Address Stack

In directives that transfer control to some other portion of the program, but will return at a later time, a special system component known as the Return Address Stack retains the address of the statement to return to. The Return Address Stack is set by the FOR/NEXT and GOSUB directives. It is also set when the Escape key has been depressed and the SETESC directive has been set (this is effectively a special type of GOSUB).

The Return Address Stack can hold more than one address; this allows for the nesting or placement of one of these returning program structures within another. This allows greater programming flexibility, but care must be taken in setting up these structures in order to avoid open, unfinished or confused processing sequences

that either do not return to the proper point or that generate execution errors.

Examples of Proper and Improper Loop and Subroutine Processing

- | 1. Loops: | Proper | Improper |
|-----------|-------------------|-------------------|
| | 10 FOR X= 1 TO 5 | 10 FOR X= 1 TO 5 |
| | 20 FOR Y= 3 TO 76 | 20 FOR Y= 3 TO 76 |
| | : | : |
| | 40 NEXT Y | 40 NEXT X |
| | 50 NEXT X | 50 NEXT Y |

Loops must be completely processed within one another; if they are nested, processing must not "cross".

2. Subroutines

10 GOSUB 100

100 -----

300 GOSUB 400

350 RETURN

400 -----

500 GOSUB 600

550 RETURN

600 -----

700 RETURN

For this series of program statements, in which a subroutine calls another subroutine, the RETURN statements at the end of the subroutine direct processing to return to the statement that called that subroutine. The 700 RETURN returns to the 500 GOSUB, the 550 RETURN to the 300 GOSUB and the 350 RETURN to the 10 GOSUB. If a RETURN statement exists, for which there is no active GOSUB to return to, an execution error is generated.

3. Mixed Loops and Subroutine Errors

100 FOR X= 1 TO 5

120 GOSUB 200

:

200 NEXT X

The NEXT directive specifies that an address to return to is obtained from the Return Address Stack, but the last setting was done by the GOSUB directive rather than a FOR directive. The result of this is that the GOSUB remains open and the NEXT will attempt to return to it rather than to the proper FOR directive; an execution error will thus be generated.

100 GOSUB 1000	The RETURN directive attempts to return
:	to the return address set by the still open
1000 FOR X = 1 TO 5	FOR/NEXT loop rather than the proper
1050 RETURN	GOSUB directive; this generates an
1060 NEXT X	error.

Directives

Program Control Directives

EXITTO	ON GOTO
FOR/NEXT	RETRY
GOSUB	RETURN
GOTO	SETERR
IF/THEN/ELSE	SETESC
ON GOSUB	WAIT

NOTE: Program control is also affected by Error Branching (see ERR=, DOM=, and END= options) and Public Programs (see Chapter 7).

FUNCTION

This directive provides an exit from a FOR/NEXT loop or a GOSUB subroutine by unconditionally branching to the specified statement number. This is the proper method of leaving an uncompleted loop or subroutine because certain program parameters are altered so that the loop or subroutine is treated as if it had been completed or "closed".

This completion is necessary to prevent later loops or subroutines from developing execution errors as they try to return to the still "open" loop or subroutine. This type of execution "confusion" error could occur if a GOTO directive, which does not complete or "close" the loop or subroutine, was used instead of an EXITTO directive.

SYNTAX

EXITTO statement-number

DESCRIPTION

- This directive must be positioned in, and branch to, a statement outside a loop or subroutine.
- For nested loops or subroutines, this directive only "closes" the loop or subroutine in which it was positioned.
- This directive can be used in Program Mode or with the EXECUTE directive only.

EXAMPLES**EXITTO 8976**

branches to statement 8976 and "closes" or completes the loop or subroutine in which it was located

IF X = 405 THEN EXITTO 8976

the positioning of the EXITTO directive within an IF/THEN directive has the same effect as example 1, but it will only be executed if X = 405

FUNCTION This directive establishes a repetitive loop for executing a sequence of program statements.

SYNTAX FOR num-variable = start-value TO end-value
[STEP step-size]

NEXT num-variable

where:

num-variable is a numeric variable name

start-value is a numeric value that is assigned as the initial value of the variable

end-value is a numeric value that is the value of the variable at which the loop will be completed

step-size is a numeric value that is used to increment the value of the variable

- DESCRIPTION**
- A numeric value is specified as the counter to control the number of times the loop is executed. The initial value, an ending value, and the size of the steps between the start and end values may be specified.
 - If a step value is not specified, the value assumed is 1. Note: An error occurs if the step value is 0.
 - The counter can be set up to go through the loop in a decreasing manner by setting the start value higher than the end value and using a negative step value.
 - FOR/NEXT loops may be nested by placing one completely inside another.
-

- Leaving a loop before it is completed may result in execution errors unless a proper GOSUB or EXITTO directive is used.
- A FOR/NEXT loop is always executed at least once because the incrementing and testing for the end value is done at the NEXT directive. Thus, the final value of the numeric variable will be greater than the end value specified for the counter.
- If an error occurs due to an illegal upper or lower limit or other condition, it is sensed at the NEXT and the loop will have been executed once incorrectly.
- This directive can be used in Program Mode only.

EXAMPLES

FOR X = 1 TO 5

NEXT X

causes the statements between the FOR and NEXT statements to be executed 5 times for the values of X = 1,2,3,4, and 5

FOR X = 10 TO 2 STEP - 3

NEXT X

causes the statements between the FOR and NEXT statements to be executed 3 times for the values of X = 10,7 and 4

FOR A = 1 TO B

FOR X = 1 TO 5

NEXT X

NEXT A

if B = 4, causes inner loop to be executed completely for each step in the outer loop

FOR X = 1 TO 5

X = 5

NEXT X

this loop will terminate after the first loop since the variable used for the counter was altered and set to its ending value

FUNCTION This directive unconditionally branches to a subroutine.

SYNTAX GOSUB statement-number

DESCRIPTION

- Program execution resumes at the statement following the GOSUB directive when the subroutine is completed. Completion of a subroutine is accomplished with the RETURN directive.
- Subroutines may be nested by placing a GOSUB directive within another subroutine, but the nested subroutines must have properly organized endings or exits or an error may be produced.
- To avoid possible execution errors, a subroutine should be terminated with a RETURN directive. An EXITTO directive should be used if a branch is made from a subroutine before it is completed.
- This directive can be used in Program Mode or with the EXECUTE directive only.
- A conditional branch may be achieved using the ON GOSUB or an IF/THEN directive that includes a GOSUB.

EXAMPLES**GOSUB 8976**

causes the program to start execution of the subroutine at statement number 8976; when a RETURN directive is reached, execution will be returned to the statement following the GOSUB directive

IF X = 8 THEN GOSUB 8976

the placement of the GOSUB directive in an IF/THEN directive has the same effect as the first example, but is only executed if X = 8

GOTO Directive

GOTO

FUNCTION This directive causes an unconditional branch or transfer of program execution to the statement specified.

SYNTAX GOTO statement-number

DESCRIPTION • A conditional branch may be achieved using the ON GOTO directive or an IF/THEN directive that includes a GOTO.

EXAMPLES GOTO 7687
causes statement 7687 to be the statement that is executed next

IF Y\$ = "YES" THEN GOTO 7687
the GOTO directive within the IF/THEN directive has the same effect as the first example, but is only executed if Y\$ = "YES"

FUNCTION This directive conditionally executes a directive.

SYNTAX IF expression [THEN] directive [ELSE directive]

where:

expression is a relational and/or logical expression used to test a true or false condition

directive (THEN) is the directive to execute if the expression is true

(ELSE) is the directive to execute if the expression is false

DESCRIPTION

- An expression is used to establish a condition that can be tested to determine if it is true or false. The expression itself is a relational condition that may be formed from other logically connected relational conditions.

- The syntax for the relational condition is:

num/str-expr relational-operator
num/str-expr

where:

num/str-expr a numeric or string expression that is related to the expression on the other side of the operator

relational-operator	= (equal to)
	< > (not equal to)
	> (greater than)
	< (less than)
	= > or > = (greater than or equal to)
	< = or = < (less than or equal to)

- Logical constructions of relational conditions can be made with the AND and OR logical operators. Any number of logical relations can be established.
- If continuation statements are constructed with this directive, care must be taken since a branch executed in an earlier portion of the directive will prevent execution of later portions.
- IF/THEN/ELSE directives may be nested or strung together to form complex logical evaluations.

EXAMPLES

IF X = 4 THEN Y = 7

constructs a relation, $X = 4$, that is tested. If the relation is true, Y is assigned the value 7. If the relation is false, execution proceeds to the next statement.

IF Y\$ = "T" OR X\$ = "U" THEN GOTO 300 ELSE PRINT "WRONG ENTRY"

if the relational conditions $Y\$ = "T"$ or $X\$ = "U"$ are true, program execution branches to statement 300. If neither one of these relations is true, "WRONG ENTRY" is printed and execution proceeds to the next statement

IF X > 7 THEN GOTO 456 ;GOSUB 567

if the condition $X > 7$ is true, execution branches to statement 456 and the continua-

tion GOSUB 567 is not executed. If the condition is false, the rest of the statement is executed and the GOSUB directive is processed

IF A = 5 THEN IF B = 6 THEN GOTO 30 ELSE
GOTO 40 ELSE GOTO 60

in this directive, the GOTO 30 is executed if A = 5 and B = 6, GOTO 40 is executed if A = 5 and B is not equal to 6 and GOTO 60 is executed if A is not equal to 5

FUNCTION This directive provides for the conditional transfer of program execution to one of the series of subroutines starting at the statements listed.

SYNTAX ON x GOSUB statement-number-list

where:

x is an integer expression or value

stmt-no-list is a list of the first statement number of a series of subroutines to branch to depending upon the value of x

DESCRIPTION

- The statement to be executed is determined as follows:

branch to 1st subroutine if

x is negative or zero

branch to 2nd subroutine if

x = 1

branch to 3rd subroutine if

x = 2

branch to nth subroutine if

x = n - 1

branch to last subroutine if

x is greater than or equal to the number of subroutines listed

- This directive has the same effects and conditions as the GOSUB directive.
- This directive can be used in Program Mode only.

EXAMPLES

ON N GOSUB 100,200,300

if $N = -2$ execute statement 100

$N = 0$ 100

$N = 1$ 200

$N = 2$ 300

$N = 7$ 300

$N = 3.4$ an error will result since N is not
an integer

FUNCTION This directive provides for the conditional transfer of program execution to one of the series of statements listed.

SYNTAX ON x GOTO statement-number-list

where:

x is an integer expression or value

stmt-no-list is a list of statement numbers to branch to depending upon the value of x

DESCRIPTION • The statement to be executed is determined as follows:

branch to 1st subroutine if

x is negative or zero

branch to 2nd subroutine if

x = 1

branch to 3rd subroutine if

x = 2

branch to nth subroutine if

x = n - 1

branch to last subroutine if

x is greater than or equal to the number of statements listed

EXAMPLES

ON N GOTO 100,200,300

if N = -2 execute statement 100

N = 0 100

N = 1 200

N = 2 300

N = 7 300

N = 3.4 an error will result since N is not an integer

FUNCTION This directive transfers program execution from an error branch taken by a SETERR directive or the ERR=, END= or DOM= options back to the statement that generated the error.

SYNTAX RETRY

DESCRIPTION

- If the error branch was controlled by a SETERR directive, this directive will reset the line number specified by the SETERR.
- This directive can be used in Program Mode or with the EXECUTE directive only.

EXAMPLES

```
0010 SETERR 100
0020 READ (2)A
0030 PRINT A
```

```
-----
0100 PRINT "ERR=",ERR
0110 RETRY
```

if an error occurs during the execution of statement 20, a branch to statement 100 is directed by the SETERR directive and the RETRY directive returns execution to statement 20 and resets the SETERR statement number to 100

```
20 READ (2,ERR = 100)A
```

```
-----
100 PRINT "ERR=",ERR
110 RETRY
```

has the same effect as the first example, but the branch is directed by the ERR= option and the RETRY directive returns execution to statement 20

FUNCTION This directive terminates a subroutine and returns program execution to the statement following the originating GOSUB or ON GOSUB directive.

SYNTAX RETURN

DESCRIPTION

- This directive also returns program execution to the statement following the point at which the Escape Key was activated if a SETESC directive has been set. The SETESC directive serves effectively as a specialized GOSUB.
- Nested GOSUB directives require a separate RETURN for each GOSUB.
- A RETURN that is not preceded by an active GOSUB or ON GOSUB will produce an error.
- A non-standard exit from an unfinished subroutine may be accomplished using the EXITTO directive.

EXAMPLES

```
0010 GOSUB 500
```

```
0011 PRINT X
```

```
-----
```

```
0500 LET X = X**Y
```

```
0501 RETURN
```

program execution branches to the subroutine at statement 500 and, when completed, the RETURN directive transfers execution back to statement 11

FUNCTION This directive transfers program execution to a specified statement if an execution error occurs that is not handled by an error branching option (i.e., ERR = , DOM = , END =).

SYNTAX SETERR statement-number

DESCRIPTION

- A RETRY directive, placed after the statement that is specified by the SETERR directive, will transfer execution back to the statement that originated the error and reset the SETERR statement number.
- When a SETERR is executed, the SETERR is automatically set to SETERR 0 and the previous SETERR value and the statement that developed the error are saved in case the RETRY directive is used. Any further errors that develop will be handled normally (return to console mode) unless another SETERR is specified. If an error branch (ERR = , END = , DOM =) is specified within the routine specified by the SETERR directive, the previous SETERR and RETRY addresses will be lost if any of these error branches are taken.
- This directive must appear in the program before any errors it is expected to handle.
- SETERR is reset to 0 when a BEGIN, CLEAR, LOAD, RESET or RUN directive is processed.

EXAMPLES SETERR 100
if an ERROR occurs that is not handled by an error option program, execution branches to statement 100

FUNCTION This directive transfers program execution to the specified statement number if the Escape key is depressed. The statement being processed when the key was depressed is completed.

SYNTAX SETESC statement-number

DESCRIPTION

- The occurrence of a RETURN directive will transfer program execution back to the statement following the statement that was being executed when the Escape key was depressed.
- This directive is not cleared or reset to 0 when it is executed, so care must be taken to prevent the possibility of starting an "inescapable loop". A SETESC 0 will return the normal action of the Escape key.
- This directive is reset to 0 when a CLEAR, BEGIN or RESET directive is processed.
- There is no relationship between the SETESC and ESCAPE directives.

EXAMPLES SETESC 100
if the Escape Key is depressed at any time during the program, execution is transferred to statement 100

WAIT Directive

WAIT

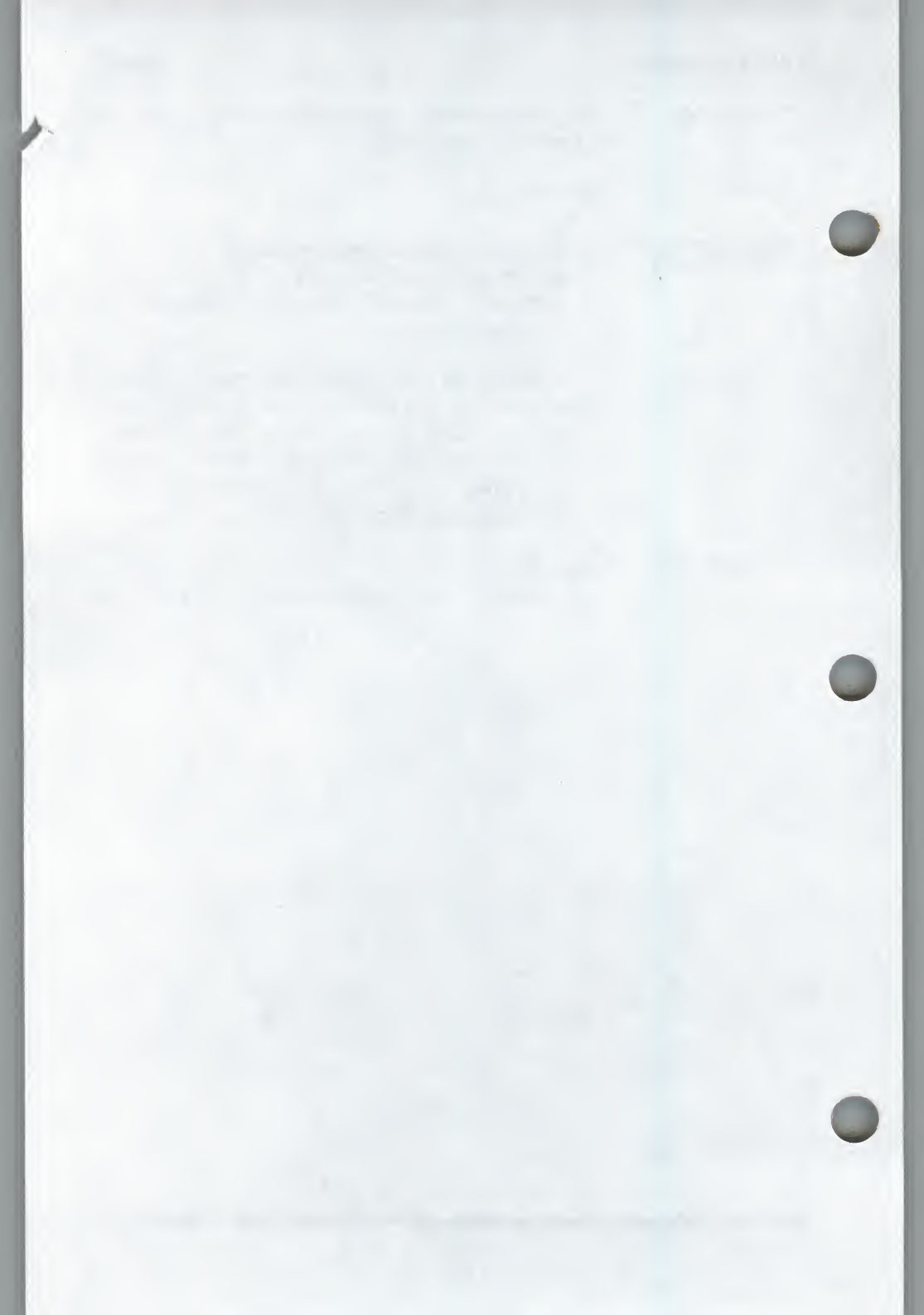
FUNCTION This directive suspends program execution for a specified period of time.

SYNTAX WAIT seconds

DESCRIPTION

- The time period specified must be an integer value between 0 and 32,767, equal to the number of seconds execution is desired to be suspended.
- A setting of 0 suspends execution for .5 seconds (the next time cycle). If the wait occurs in the middle of a time cycle, the delay may be shorter than specified. Note: Certain machines may differ in the amount of time the execution is suspended.

EXAMPLES WAIT 23
suspends program execution for 23 seconds



CHAPTER FIVE—DISK AND FILE OPERATIONS

Computer operations must depend upon some facility for the storage of information or data. Two types of storage facilities are provided with Thoroughbred/OS:

- **Random Access Memory (RAM)**

This memory system is utilized by the CPU for storing data and programs that are currently being processed. RAM memory is dynamic and constantly changing as Tasks and programs are being processed.

- **Mass Memory**

Mass Memory is used for the permanent or long term storage of information such as programs, or data files. Data in mass storage is usually located on a physical medium such as disks or tapes.

This chapter deals with the use of Mass Memory storage of data. This includes logical disks, disk access, files, and file creation and access. The actual process of transferring information into and out of mass memory is discussed in the chapter on Input/Output processing. Aspects of the management and use of RAM memory are discussed in the chapter on System Directives.

DISKS

Within Thoroughbred/OS are special memory units referred to as Logical Disk volumes. A logical disk is a subset of the entire physical storage system on which information or data, in the form of files, is kept. This serves to divide the total memory capacity into convenient blocks. The designation of a logical disk, its capacity and its contents is made through the operating system.

Disk Accessing and Restricting Directives

DISABLE
ENABLE
RESERVE

FILES

Within memory, data is arranged in convenient groupings called files. Each file consists of a collection of related data that is divided into records that contain data groups usually related to a single item. Within records are further divisions, called fields, that contain distinct data items. There are two types of files supported by Thoroughbred/OS: program files and data files. Within the data file classification are Indexed, Direct and Sort files.

File Types

Program files

- store Thoroughbred/OS programs
- each record in the file contains a program statement or the continuation of a statement
- program files have certain restrictions on how they may be handled, (i.e., they may not be input or output in the same manner as data files with I/O directives)
- a program is handled in its entirety when transferring it to or from RAM memory
- program size cannot be larger than the total of RAM memory. In situations where large programs are needed, they may be divided into segments or overlays, each a separate program that will execute, at its completion, the next overlay

Indexed files

- store data in file records
- file records are identified by Index number, this number being the sequential position or number of the record within the file
- each record is of a specified length and may be accessed either sequentially or randomly by Index value

Direct files

- store data in file records
- file records are identified by Index number and Key value, the Index number corresponding to its sequential position in the file and the Key value being a unique string value that corresponds and is associated with the Index number of the record
- each record is of a specified length and may be accessed either sequentially or randomly by Key or Index values

Sort files

- contain no data, only a Key value for the record
- Sort files are used as convenient means of sorting data or accessing records in a sorted fashion in Direct or Indexed files

For a discussion of Key and Index values and their use in accessing records, refer to Input/Output Processing. For a discussion of programs and their structure, refer to the chapter on System Directives.

File Creation

Files are created using a series of file directives that instruct the system to assign storage space for the file. An entry in the Disk Directory is also made that records the name of the file and its parameters (i.e., number and size of records, and size of the Key, etc.). All files have a name that is one to eight characters in length and is unique in the system. It may not contain any space (" ") or null (\$00\$) characters. The following two character names are reserved for device and Task names; avoid using them for file names:

"LP"	primary printer	"SY"	Synchronous Communication
"P1"		"T0"	
:	additional printers	:	User Tasks
"P9"		"TF"	
"D0"		"G0"	
:	Disk Volumes	:	Ghost Tasks
"D9"		"GF"	

File Creation/Erase Directives

DIRECT
ERASE
FILE
INDEXED
PROGRAM
RENAME
SAVE
SORT

File Accessing

Files, except in the case of Program files which are transferred in their entirety, must be handled in discrete units of data or records. The file remains in mass memory, but the record of data is transferred into RAM memory. The OPEN directive establishes a channel through which files are accessed. The OPEN directive establishes an entry in the File Control Table for that file or device that holds disk location information that speeds later accessing. In addition, there are directives used to control access to files.

This method of access is also applied to Input and Output devices such as terminals and printers.

File Accessing Directives

ADD
CLOSE
DROP
LOCK
OPEN
UNLOCK

FUNCTION

This directive establishes an entry for a file in the File Control Table. This table contains information on the parameters and disk storage location of files. This speeds up later access, since a time consuming disk search is not required to find these parameters.

SYNTAX

ADD file-id [,ERR = stmt-no]

where:

file-id is a string value specifying a file name

stmt-no is the number of the statement to branch to if an error is produced by this directive

DESCRIPTION

- The number of entries allowed in this table is controlled by system considerations and the type and status of other entries in the table.
- Program files may also be entered into this table.
- Files that are added to the table may be removed with the DROP directive.

EXAMPLES

ADD "INDEX"

makes an entry for the file "INDEX" in the File Control Table

ADD A\$,ERR = 7999

if A\$ = "INDEX", has the same effect as the first example; in addition, will branch to statement 7999 if the directive produces an error condition

FUNCTION This directive releases an OPEN file or device and its assigned channel number.

SYNTAX CLOSE (channel [,ERR = stmt-no])

where:

channel is an integer value specifying the channel of an OPEN file or device

stmt-no is the number of the statement to branch to if this directive produces an error condition

- DESCRIPTION**
- A file or device remains OPEN and its assigned channel unavailable for other use until it is closed.
 - An OPEN device is unavailable to another Task until closed.
 - All OPEN files and devices in a Task are closed by the BEGIN, END, START or STOP directives.
 - The entry for the file is retained in the File Control Table even after it is closed to speed up later access. The entry is only removed if the system requires the space or the file is dropped.

EXAMPLES

CLOSE (1)
closes the file or device OPEN on channel 1

CLOSE (1,ERR = 7999)
has the same effect as the first example and, in addition, will branch to statement 7999 if the directive produces an error condition

FUNCTION This directive defines the parameters of a Direct file and makes an entry in the Disk Directory for the name and parameters of the file.

SYNTAX DIRECT file-id, key-size, no-records,
record-size, disk-no, sector-no
[,ERR = stmt-no]

where:

file-id is a string value specifying a file name

key-size is an integer value indicating the size (the number of bytes) of the file's Key value

no-records is an integer value indicating the number of records in the file

record-size is an integer value indicating the number of bytes in a record

disk-no is an integer value indicating the logical disk on which to store the file

sector-no is an integer value indicating the starting sector number for the storage location on the disk

stmt-no is the number of the statement to branch to if this directive produces an error condition

DESCRIPTION • The specification of sector number 0 directs the system to handle space allocation.

EXAMPLES DIRECT "SEAL", 10, 57, 26, 2, 200
creates a Direct file named "SEAL" with the

following parameters: Key length is 10 bytes,
57 records with a length of 26 bytes each,
and locates it on disk 2 starting at sector 200

DIRECT A\$, A, B, C, D, E, ERR = 7999

if A\$ = "SEAL", A = 10, B = 57, C = 26, D = 2,
and E = 0 has the same effect as the first
example, but directs the system to allocate
space for the file and, in addition, branches
to statement 7999 if the directive produces
an error condition

FUNCTION This directive prevents access to a specified logical disk. This protects files from damage or secures them from unauthorized access.

SYNTAX `DISABLE disk-no [,LOCAL]`

where:

`disk-no` is an integer value indicating the logical disk

`LOCAL` is an optional modifier that limits the effect of this directive to the issuing Task

- DESCRIPTION**
- Only the Task that issued the directive can change the **DISABLE** status by issuing the **ENABLE** directive.
 - The effect of this directive ends if the issuing Task is terminated by the **RELEASE** directive.
 - If an attempt is made to **DISABLE** a disk which still has open files or is already disabled, an error will be produced.

EXAMPLES `DISABLE 2`
removes the ability of any Task to access disk 2

`DISABLE A, LOCAL`
if `A=2`, has the same effect as the first example but only prevents access to disk 2 from the Task that issued this directive

FUNCTION

This directive removes an entry for a file from the File Control Table that has been made with the ADD or ADDR directives.

SYNTAX

DROP file-id [,ERR = stmt-no]

where:

file-id is a string value specifying a file name

stmt-no is the number of the statement to branch to if this directive produces an error condition

DESCRIPTION

- An error is produced if an attempt is made to DROP a file that is OPEN on any Task.

EXAMPLES

DROP "INDEX"

removes the entry for the file "INDEX" in the File Control Table

DROP A\$,ERR = 7999

if A\$ = "INDEX", has the same effect as the first example and, in addition, will branch to statement 7999 if an error condition is produced

FUNCTION This directive allows access to a specified logical disk by either adding it to the system configuration, or ending the effects of a previous DISABLE directive.

SYNTAX `ENABLE disk-no [,LOCAL]`

where:

`disk-no` is an integer value specifying a logical disk

`LOCAL` is an optional modifier that limits the effect of this directive to the issuing Task

DESCRIPTION

- An attempt to ENABLE a disk that is already enabled or one that was disabled by another Task will produce an error.

EXAMPLES

`ENABLE 3`
adds logical disk 3 to the system configuration, or removes the effect of a DISABLE directive on the disk

`ENABLE A, LOCAL`
if `A=3`, has the same effect as the first example but limits the effect to the Task that issued the directive

FUNCTION This directive effectively erases a file by removing the Disk Directory entry for the file. This entry contains information on its disk location and defined parameters.

SYNTAX ERASE file-id [,ERR = stmt-no]

where:

file-id is a string value that specifies a file name

stmt-no is the number of the statement to branch to if this directive produces an error condition

- DESCRIPTION**
- This directive applies to both data and program files.
 - While the Disk Directory entry is removed by this directive, data within the file will remain intact until it is altered by a later operation on that physical area of the disk. This allows possible recovery of the information by redefining the file with the PROGRAM, INDEXED, or FILE directives. This is done by redefining the file with the same parameters at the same location.
 - Attempts to erase a file that is open on another Task will produce an error.

EXAMPLES

ERASE "INDEX"
erases the file "INDEX"

ERASE A\$, ERR = 7999
if A\$ = "INDEX", has the same effect as the first example and, in addition, branches to statement 7999 if the directive produces an error condition

FUNCTION

This directive defines a file using a string in the format of the FID variable that contains the file parameters. The file is defined and an entry is made in the Disk Directory for its name and parameters.

SYNTAX

FILE string-value

where:

string-value is a string value, in the format of the FID variable, that contains the file parameters

DESCRIPTION

- The format of the FID variable is a 22-byte string as follows:

Bytes	1-3	Starting sector (hexadecimal)
	4-9	File name (characters 1-6),
	10	Coded file type: 0—Indexed file 2—Direct or Sort file 4—Program file
	11	Key size + Pointer size (hexadecimal) (pointer size = 4, less than 32,768 records) (pointer size = 6, more than 32,768 records)
	12-14	Number of records (hexadecimal)
	15-16	Bytes per record (hexadecimal)
	17-19	Ending sector + 1 (hexadecimal)
	20	Logical disk number (hexadecimal)
	21-22	File name (characters 7-8),

- This method of file definition can be used to recover erased files if the contents of the FID variable for the erased file are still retained. It may also be used to define files with parameters similar to a pre-existing file.
- In order to restore Direct or Sort files without clearing the Key area, bit 10 of the string must be set to \$40\$.

EXAMPLES

FILE A\$

if $A\$ = FID(1)$, redefines the file (previously OPEN on channel 1 and then erased) with the same parameters; if the memory area of the file has not been over-written since the file was erased, this may be used to restore the file with its previous contents intact

FILE B\$

if $B\$ = \$000000\$ + \text{"INDEX"} + A\(10) and $A\$ = FID(1)$, creates a file with the same parameters as the file OPEN on channel 1, names it "INDEX", and directs the system to locate it on the disk

FUNCTION

This directive defines the parameters of an Indexed file and makes an entry in the Disk Directory for the name and parameters of the file.

SYNTAX

INDEXED file-id, no-records, record-size,
disk-no, sector-no [,ERR = stmt-no]

where:

file-id	is a string value specifying a file name
no-records	is an integer value specifying the number of records in the file
record-size	is an integer value specifying the number of bytes in a record
disk-no	is an integer value indicating the logical disk on which to locate the file
sector-no	is an integer value specifying the starting sector number for locating the file
stmt-no	is the number of the statement to branch to if this directive produces an error condition

DESCRIPTION

- Specifying a sector number of 0 directs the system to handle the assignment of storage space.

EXAMPLES

INDEXED "SEA3", 75, 25, 2, 100
creates an Indexed file with 75 records, with a length of 25 bytes each, and assigns it to be stored on disk 2 starting at sector 100

INDEXED A\$, A, B, C, D, ERR = 7999

if A\$ = "SEA3", A = 75, B = 25, C = 2, D = 0, has the same effect as the first example, but instructs the system to assign the sector address for storage and, in addition, will branch to statement 7999 if an error condition occurs

LOCK Directive

LOCK

FUNCTION

This directive allocates exclusive use of a file to the Task that issued the directive.

SYNTAX

LOCK (channel [,ERR = stmt-no])

where:

channel is an integer value that specifies the channel assigned to an OPEN file

stmt-no is the number of the statement to branch to if this directive produces an error condition

DESCRIPTION

- The directive may only be applied to a file that has already been opened by the Task and is not currently open on any other Task.
- A file may only be locked in an open state.
- This directive is cancelled by the UNLOCK or CLOSE directive or by a BEGIN, END, START or STOP directive.
- This directive may not be applied to a terminal or device.

EXAMPLES

LOCK (1)

reserves the file OPEN on channel 1 for the exclusive use of the Task that issued the directive

LOCK (A,ERR = 7999)

if A = 1, has the same effect as the first example and, in addition, will branch to statement 7999 if the directive produces an error condition

FUNCTION

This directive directs access to a file or device through an assigned channel number, and sets the Key and/or Index values of the file record pointer to their initial values.

SYNTAX

OPEN (channel [,ERR = stmt-no]
[,ISZ = record-size]) file-id

where:

channel is an integer value specifying the number of the channel to assign to the file or device

stmt-no is the number of the statement to branch to if this directive produces an error condition

record-size is an integer value specifying the number of bytes for the temporary record size used in accessing a file

file-id is a string value specifying the name of the file or device to OPEN

DESCRIPTION

- Opening a file establishes an entry in the File Control Table containing information on the disk location of the file. This speeds accessing when later I/O operations are conducted on the file. The entry is retained even after the file is closed until system considerations require additional space in the table.
- Up to 14 files or devices, in addition to the User Task Terminal, can be OPEN at one time (this, then equals 15 devices).
- An OPEN device can only be used by the Task that issued the directive until it is closed.

- An OPEN file or device and its channel number are released by the CLOSE directive or a BEGIN, END or STOP directive.

EXAMPLES

OPEN (1) "INDEX"

assigns the channel number 1 to the file "INDEX" for use in any I/O operations

OPEN (2) "LP"

assigns the channel number 2 to the device named "LP" (printer) and restricts access by any other Task to that device until it is closed

OPEN (A,ERR = 7999,ISZ = 32)B\$

if A = 1, B\$ = "INDEX", has the same effect as the first example, and, in addition, branches to statement 7999 if the directive produces an error condition; also, assigns a temporary record size of 32 bytes to be used when accessing the file

FUNCTION This directive defines the parameters of a Program file and makes an entry in the Disk Directory for the name and parameters of the file.

SYNTAX PROGRAM file-id, size,disk-no,sector-no
[,ERR = stmt-no]

where:

file-id is a string value specifying a file name

size is an integer value specifying the number of bytes to be allocated for storage of the file

disk-no is an integer value indicating the logical disk on which to store the program

sector-no is an integer value indicating the starting number of the sector at which to store the program

stmt-no is the number of the statement to branch to if this directive produces an error condition

DESCRIPTION • The specification of sector number 0 directs the system to handle space allocation.

EXAMPLES PROGRAM "TEST", 750, 2, 100
defines a program file named "TEST" 750 bytes long, located on disk 2, starting at sector 100

PROGRAM I\$, A, B, C, ERR = 7999
if I\$ = "TEST", A = 750, B = 2, C = 0, has the same effect as the first example, but directs

the system to assign sector storage and, in addition, branches to statement 7999 if the directive produces an error condition

RENAME Directive

RENAME

FUNCTION This directive renames a file.

SYNTAX RENAME disk-no, old-file-id, new-file-id

where:

disk-no is an integer value specifying the logical disk on which the file to be renamed is located

old-file-id is a string value specifying the file to be renamed

new-file-id is a string value specifying the new file name

DESCRIPTION • This directive renames the old file and erases the disk directory reference to the old file name.

EXAMPLES RENAME 2, "DCINDX", "INDXPM"
changes the name of the file "DCINDX" on disk 2 to "INDXPM"

RENAME X, A\$, B\$
if X = 2, A\$ = "DCINDX" and B\$ = "INDXPM"
has the same effect as the first example

RESERVE Directive

RESERVE

FUNCTION Restricts logical disk access.

SYNTAX **RESERVE** disk-no

where:

disk-no an integer value specifying a logical disk

- DESCRIPTION**
- This directive restricts accessing of a logical disk to the Task that issued the directive (valid for most operating systems). The directive serves to remove the logical disk from the system configuration of all Tasks except the issuing one.
 - A disk may not be reserved if other Tasks are accessing it and have files OPEN on that disk or if it is already reserved. In these cases an error will be produced.
 - **RESERVE** remains in effect until the Task that issued the directive issues an **ENABLE** directive for that disk or the Task is terminated with the **RELEASE** directive.

EXAMPLES **RESERVE X**
if X = 4, restricts access to logical disk 4 to the Task that issued the directive

SAVE Directive

SAVE

FUNCTION

This directive saves a program by storing it on disk or tape in the form of a Program file. If the program has not already been defined, this directive also is used to define the program file.

SYNTAX

SAVE [prog-id] [,size, disk-no, sector]
[,ERR = stmt-no]

where:

prog-id is a string value specifying the name of the program file that the program will be saved under

size is an integer value specifying the size of the program (number of bytes)

disk-no is an integer value indicating the disk on which to store the program

sector is an integer value specifying the starting sector number for the storage of the program

stmt-no is the number of the statement to branch to if this directive produces an error condition

DESCRIPTION

- Saved programs are written over previous copies of the program and remain in the Task area after the SAVE operation is complete.
- Specification of the optional prog-id is used to store the current program at a previously defined location. If the prog-id is not specified, the program is saved under the name of the current program in the Task area.

- The optional program size and disk and sector numbers are used when defining a Program file for the first time.
- The specification of sector number 0 directs the system to handle space allocation.

EXAMPLES

SAVE

saves the program currently in the Task area under the same name as the current program and replaces the previous copy

SAVE "TEST"

saves the current program in the space defined for the previously defined program "TEST"

SAVE A\$, 100, 2, 0, ERR = 7999

if A\$ = "TEST1", the current program is defined as the Program file "TEST1" with a size of 100 bytes and is stored on disk 2 at a starting sector determined by the system. In addition, will branch to statement 7999 if the directive produces an error condition

FUNCTION This directive defines a Sort file and makes an entry in the Disk Directory for the name and parameters of the file.

SYNTAX SORT file-id, key-size, no-keys, disk-no,
sector-no [,ERR = stmt-no]

where:

file-id is a string value specifying a file name

key-size is an integer value specifying the size (number of bytes) of the Key value

no-keys is an integer value specifying the number of Key values in the file

disk-no is an integer value indicating the disk on which to store the file

sector-no is an integer value specifying the starting sector location for storage of the file

stmt-no is the number of the statement to branch to if this directive produces an error condition

DESCRIPTION

- The specification of sector number 0 directs the system to handle space allocation.
- In a Sort file there is no data area as such since there are no records, only Key values.

EXAMPLES SORT "FIFE", 15, 200, 2, 450
defines a Sort file named "FIFE" with a 15-byte Key size and 200 Key values and stores it on disk 2 starting at sector 450

SORT S\$, A, B, C, D, ERR = 7999

if S\$ = "FIFE", A = 15, B = 200, C = 2, D = 0,
has the same effect as the first example but
directs the system to allocate sector storage
and, in addition, branches to statement 7999
if the directive produces an error condition

FUNCTION

This directive removes the LOCK access restriction on a file.

SYNTAX

UNLOCK (channel [,ERR = stmt-no])

where:

channel is an integer value that specifies the assigned channel number of an OPEN file

stmt-no is the number of the statement to branch to if this directive produces an error condition

DESCRIPTION

- A file that has been locked cannot be accessed by another Task until it is unlocked by this directive or until a BEGIN, END, START or STOP directive is processed.
- This directive does not close the file or release the assigned channel number.

EXAMPLES

UNLOCK (3)

removes the LOCK access restriction of the file OPEN on channel 3

UNLOCK (A, ERR = 7999)

if A = 3, has the same effect as the first example, and, in addition, will branch to statement 7999 if the directive produces an error condition

CHAPTER SIX—INPUT/OUTPUT PROCESSING

In most computer applications, data must be transferred from one component of the system to another. Data transferring capabilities in Thoroughbred/OS are provided by a number of Input/Output (I/O) directives. There are a number of options and accessory capabilities related to I/O that greatly increase the utility of this process. These include the ability to test and verify input data, format the output of numeric values, position output on a terminal or printer, and control printer and terminal actions. This chapter discusses both the directives and related I/O capabilities.

INPUT/OUTPUT

Input involves the process of getting information into a program. The source of this information may be either a file or a terminal. Output is the process of transferring data from a program onto a storage media such as tape or disk, or printing program results at a terminal or printer.

Thoroughbred/OS has a "sector-cacheing" feature for all disk I/O operations. This feature allows the system to use a portion of Random Access Memory to minimize physical disk accesses to contiguous disk sectors. See the *NPSD Utility program description in Chapter 9 for details on how to set the sector-cacheing parameters. The following I/O directives are documented in this chapter:

Input Directives

EXTRACT

FIND

GET

INPUT

READ

Output Directives

PRINT

PUT

WRITE

Data Removal Directives

REMOVE

Definition Statements

IOLIST

TABLE

I/O DIRECTIVE AND OPTION CAPABILITIES

This section deals with the capabilities of the I/O directives and options that are provided. It is divided into the following sections:

- Record Specification
 - KEY = I/O option
 - IND = I/O option
- Error Branching
 - ERR = I/O option
 - END = I/O option
 - DOM = I/O option
- Input/Output List
 - IOLIST statement
 - IOL = I/O option
- Code Conversion
 - TABLE statement
 - TBL = I/O option
- Miscellaneous I/O Options
 - TIM = I/O option
 - SIZ = I/O option
 - LEN = I/O option
- Input Verification
- Data Positioning
- Output Formatting
- Device Control
- Special Characters

Record Specification

KEY = specifies a Key value

IND = specifies an Index value

Data is manipulated in units of file records when being input to, or output from, a file. All I/O operations dealing with files access only one record at a time, unless the ISZ = option of the OPEN directive is used to assign a temporary record size other than the defined record size. The record to be accessed is indicated by values that are maintained in the record pointer for that file. This record pointer holds the Index value of a record for Indexed files, and the Index and Key values of a record for Direct and Sort files.

An Index value for a record is a unique number that is associated with one record in the file. A Key value is a unique string value that is associated with one record and is used to establish a means of sorting the records in the file according to an alphanumeric sequence. The record pointer will either indicate the Key and/or Index value of the next record to be accessed, or the last record to be accessed, depending upon the last I/O directive that was executed and whether its execution was complete. For a discussion of the exact effect of the I/O directives on the Key and Index values, refer to the listing of the appropriate directive. The IND and KEY Task functions are used to determine the Index or Key values of the record pointer.

The OPEN directive initializes the record pointer to the first Key and/or Index values. The initial Index value is 0 and the initial Key value is set to whatever the initial sorted Key value is. In any subsequent I/O operations that do not specify the IND= or KEY= options, accessing is according to sequential Index values for Indexed files and sequential Key values for Direct and Sort files. In order to randomly access file records, the IND= (for Indexed files only) and KEY= I/O options are used to specify the values of the record to be accessed. The IND= and KEY= options are mutually exclusive; they cannot be used in the same directive, and they are also exclusive with the TIM= option.

Error Branching

ERR = an unspecified error branch

END = an End of File error branch

DOM = a Duplicate or Missing Key error branch

Thoroughbred/OS I/O options include three error-branching specifiers to handle errors that may develop during the processing of an I/O directive: ERR= specifies the statement number to be branched to if an error occurs during processing of the directive; END= specifies the statement to branch to if an END FILE error (Error 2) develops; DOM= specifies the statement to branch to if a Duplicate or Missing Key error (Error 11) develops.

When more than one of these three options is present in a directive, an order of precedence exists as follows: a) the occurrence of either the END= or DOM= options is processed before the ERR= option; b) if none of the three options are present, and the I/O direc-

tive develops an error, it will be processed by a SETERR directive; c) if a SETERR directive is not present, the program will suspend execution. Note: The END = and DOM = options are mutually exclusive; they cannot appear in the same directive.

Input/Output List

IOLIST—defines the content of an I/O list
IOL = specifies the location of an IOLIST

For I/O directives in which the list of variables and input or output modifiers is long or is repeated in many different directives, it may be convenient to use the IOLIST statement to define the list. This list is then referred to by the IOL = option that refers to the statement number at which the IOLIST is located. The IOLIST definition statement can be used to define variable lists and may also be used to define mnemonics (strings to be printed as output, and input verification). It can be used in all of the I/O directives except the REMOVE directive and RECORD modified directives (i.e., READRECORD). The allowable components of the IOLIST depend upon the directive in which it appears.

Code Conversion

TABLE—defines a code conversion table
TBL = specifies the location of a TABLE statement

Occasionally, input or output will need to be transformed from one code system to another, or certain code characters may need to be converted into different characters. This is accomplished by developing a code conversion table with the TABLE definition directive, and by using the TBL = I/O option to refer to the statement at which the table is located in order to conduct the transformation.

Miscellaneous I/O Options

TIM = specifies an input time limit
SIZ = specifies an input length
LEN = specifies an input length

Three I/O options can be used with the INPUT directive. These are: the TIM = option, for specifying a time limit in which to enter data; the SIZ = option, that specifies the number of characters to enter before an automatic 'CR' (carriage return) is executed; the LEN = option, that allows input of only a certain length.

The TIM = option is used to specify the number of seconds (an integer value from 0 to 32767) that is allowed for the entering of data. If the allowed time limit is passed, an execution error (error 0) is generated. This option is used to prevent a locking up of files, records, or other system resources since the error may be channeled to take care of these situations. This option is mutually exclusive of the IND = and KEY = options.

The SIZ = option is used to specify the number of characters allowed to be input before an automatic 'CR' is returned by the system. This can be used to speed up data entry or truncate input data to a certain number of characters. This option specifies an integer value that is the number of characters that can be entered before the automatic 'CR' is generated. When the specified number of characters is entered, the 'CR' is executed and the CTL Task variable is set to the value of 5. If a 'CR' is entered by the operator before the allotted number of characters is entered, it overrides the SIZ = option.

The SIZ = option will only affect the first variable in a variable list. If a 'CR' is to be included in the input data, the READRECORD option must be used. When the RECORD modification is used with any I/O directive, it is suggested you use the SIZ = option in order to prevent the system from hanging while waiting for a termination character. The specification of SIZ = 0 has the same effect as not specifying SIZ = .

The LEN = option is used to restrict input data to a certain length. An integer value is specified which is the maximum number of characters allowed to be entered. If an attempt is made to enter more characters than specified, a bell is sounded and the cursor is not advanced. The operator must terminate the entry with a carriage return ('CR') or control key response. If LEN = is specified, it only affects the first data element to be entered. This option should not be confused with the LEN option for input verification.

INPUT VERIFICATION

An option to the INPUT directive is provided to test data that is entered against certain conditions. The result of this testing may be used to:

- branch to a statement if a particular string is entered
- produce an error if a numeric value is entered that is outside a specified range
- produce an error if a string is entered that is not of a specified length or range of length

The errors that are produced when certain conditions are not met may be channeled, using error processing options.

Numeric Verification

The form of numeric verification is:

numeric-variable: ([-] numeric-value)

where:

numeric-value may be any constant, variable or expression and specifies the maximum value of the range (from 0) that is allowed. If a minus sign is specified, the value is evaluated to include the minimum negative value also. If the value entered is outside of the specified range, an error is produced. The setting of PRECISION may affect the results of this verification option.

Examples

A: (45)

Legal values are from 0 to 45; values less than 0 or greater than 45 will produce an error (under the default setting for PRECISION of 2, the input of 45.001 will not produce an error).

S: (- T)

If T = 23, legal values are between - 23 and 23.

Usage

INPUT (0,ERR = 200)A: (6)

If a value from 0 to 6 is entered, input is verified and the next statement is processed. If the value entered was outside of this range, an error is produced which can be handled by the ERR = option that branches to statement 200. Statement 200 may be

either a statement that handles the error, prints an explanatory remark and directs execution back to this statement, or it may be this statement itself to allow for a new entry to be attempted.

String Verification

The form of the string verification is:

string-variable: ([match = stmt-no] [,] [LEN = min, max])

where:

match is a string constant, variable or expression that is compared to the input string; if a match occurs, the program branches to the specified statement number.

LEN is used to specify the minimum and maximum length range allowed for the input string.

If a match test, but no length test is specified, and no match is made, an error is produced. If the match and length test or only the length test, are specified, an error is produced only if the input value is outside of the allowed range. Input verification can also be used when entering substrings with the INPUT directive. When more than one input verification test is used, they are processed in order from left to right.

Examples

A\$: ("TH" = 200)

If the input string is "TH", execution proceeds to statement 200.

If the input is not "TH", an error is produced.

A\$: (LEN = 2,4)

If the input string is from 2 to 4 characters in length, execution proceeds. If it is outside of this range, an error is produced.

A\$: ("TH" = 200, LEN = 4,6)

If the input string is "TH", execution proceeds to statement 200.

If the string is not "TH", it must be from 4 to 6 characters in length or an error will be produced. Note that "TH", which is only 2 characters, may be entered because the match test is made before the length test.

A\$ (4,4): ("TH**" = 200,S\$ = 400)

If S\$ = "SHII" and the input string is "TH**", then "TH**" is entered into A\$ starting at substring position 4 for a length of four characters and execution proceeds to statement 200. If "SHII" is entered, it is entered into the same position in A\$ and execution branches to statement 400; if neither of these entries is made, an error is produced; a length test cannot be made when a substring input is being verified.

Usage

INPUT (0,ERR = 200)A\$: ("TH" = 400, LEN = 2, 4)

If the input string is "TH", execution proceeds to statement 400. If the input string is not "TH", and is outside of the length range specified, an error is produced which can be handled by the ERR = option that branches to statement 200. If the input is between 2 and 4 characters in length, execution proceeds to the next statement. Statement 200 may be either a statement that handles the error, prints an explanatory remark and direct execution back to this statement, or it may be this statement itself to allow for a new entry to be attempted.

DATA POSITIONING

This option is used with both input and output and IOLIST directives when it is desired to locate printing or position the cursor or printhead when accessing a terminal or printer. Positioning is accomplished using a combination of location specifiers and mnemonic device control specifiers or special control characters. Positioning on a terminal screen can take advantage of the two dimensional screen by jumping around to print at any location. Positioning on a printer can only occur within the limits of a single line.

Position Specifiers

@(column,[row]) for terminals only

@(column) for terminals and printers

The position specifiers use a numeric constant, variable or expression that produces an integer value to locate the column (horizontal position) and/or row (vertical position) at which to start printing or position the cursor or printhead.

Mnemonic and Character Controls

- 'CR' (carriage return) is used to move the cursor or printer to its leftmost position
- 'LF' (line feed) is used to move the cursor or printhead (actually, the paper, not the printhead, moves) to the next line

Position Handling

- If no positioning specifier is used, all actions start from the current position of the cursor or printhead.
- A 'CR' produces an automatic 'LF' and a 'LF' produces an automatic 'CR'.
- Certain terminal or printer devices may produce an automatic 'CR' or 'LF' after the last character in a line.
- Each INPUT and PRINT directive produces an automatic 'CR' and 'LF', unless ended with a comma which suppresses this feature and leaves the cursor printhead at the character succeeding the last one that was printed.
- The use of a comma to suppress the automatic 'CR' and 'LF' can be used to print multiple PRINT or INPUT statements on a single line.
- The use of the "suppression comma", when directed to a printer, is specially processed in Thoroughbred/OS by saving all of the PRINT or INPUT data in a buffer until the next 'LF' or PRINT or INPUT directive without the ending comma is reached. At this point, all of the data to be printed is sorted into proper position and output. This circumvents the fact that a printhead cannot move backwards and allows data that appears later in the output line to be specified in earlier PRINT or OUTPUT statements.

Examples

PRINT @ (2,8),"ENTER",S\$

Positions at column 2, row 8, the starting character of the text to

be printed, "ENTER", and follows by printing the contents of S\$; this is followed by an automatic line feed and carriage return that brings the cursor of the terminal to the leftmost position of the next line.

INPUT @ (2,8),"ENTER ITEM:","LF","LF",@(21),X\$

Positions the start of the text "ENTER ITEM:" at column 2, row 8 and then moves the cursor down three lines where it positions the cursor for the input of X\$ at column 21. The cursor is then moved to the leftmost position of the next line by the automatic 'CR' and 'LF' produced when the directive is completed.

PRINT (2) @ (30),X\$,

Outputs at the printer, at column position 30, the contents of X\$ and suppresses the line feed and carriage return to allow any further PRINT statement to be output to the same line.

MNEMONIC DEVICE CONTROL

This I/O capability is used to accomplish certain control features on terminal and printer devices that are accessed by I/O directives, usually INPUT or PRINT. They may appear at any point in an input or output list and are effected as processed. The effect of certain mnemonics can also be achieved by certain control characters. Certain effects such as background/foreground, bells and plot mode or terminal read may be dependent upon the capabilities of the terminal, printer or system.

Terminals and Printers

'CR'
carriage return

Moves cursor or printer to the leftmost (column 0) position. For a printer, any contents of the current line in the buffer are printed before the 'CR' is effected. For a terminal, each 'CR' produces a 'LF'.

'ES'
escape

Causes an escape control character (ESC) to be transmitted. The effect is dependent upon the device.

'LF'
line feed

Causes the cursor or printhead to move to the next lower line. For a printer, any contents of the current line in the buffer are printed before the 'LF' produces a 'CR'.

'RB'
ring bell

Rings bell.

'VT'
vertical tab

Causes the cursor to move up one line or from the first line to the last line. A printer will slew to VFU channel 6.

Terminals Only

Note: Some of these mnemonics may also be transmitted to a printer without causing an ERR=29; however, they will have no effect on the printer.

'BS'
backspace

Moves the cursor one position to the left.

'CF'
clear foreground

Clears all foreground (bright) characters from the screen. Also produces a 'CH' and 'SF'.

'CH'
cursor home

Moves the cursor to the upper left most corner of the screen.

'CL'
clear line

Clears all character to the right of the cursor in the current line. The cursor and current character are not changed. Also produces a 'SF'.

'CS'
clear screen

Clears the entire screen and produces a 'CH' and 'SF'.

'LD'
line delete

Clears all characters in the current line and scrolls all lines below the current line up one line. The last line will be blank. The current cursor position is not changed. Also produces a 'SF'.

'LI'
line insert

Causes the current line and all lower lines to be moved down one line, with the last line disappearing off the screen. The cursor position is not changed and its current line remains blank. Also produces a 'SF'.

'SB'
start background

Causes all succeeding characters to be displayed in background (lower brightness) until the next mode change. The cursor position and all previous characters remain unchanged.

'SF'
start foreground

Causes all succeeding characters to be displayed in foreground (bright intensity) until the next mode change. The cursor position and all previous characters remain unchanged.

'TR'
terminal read

Causes all characters on the screen from the home position to the cursor to be transmitted to the computer.

Printers Only

'8L'
eight lines/inch

Causes the paper to advance 1/8 inch (rather than 1/6 inch which is the default value after printing the current line).

'EL'
end load (VFU)

Signals the end of VFU loading.

'EP'
expanded print

Causes the next line to be printed in expanded print.

'FF'
form feed

Causes the printer to advance to the top of the next page; also processes a 'LF'. This is not a true form feed, but produces 66 line feeds. To alter the default value of 66, consult the appendix on VFU Loading.

'PM'
plot mode

Causes the terminal to enter plot mode.

'SL' start load (VFU)	Signals the start of loading of the VFU. Used in a PRINT statement to load data that is enclosed by the 'SL' and 'EL' mnemonics. Refer to the appendix on VFU Loading.
'S2'	Slows to VFU channel 2.
'S3'	Slows to VFU channel 3.
'S4'	Slows to VFU channel 4.
'S5'	Slows to VFU channel 5.
'S6'	Slows to VFU channel 6.
'S7'	Slows to VFU channel 7.
'S8'	Slows to VFU channel 8.

User-Configured Mnemonics

In addition to the above standard preconfigured mnemonics, it is possible for the user to define new ones or alter existing ones.

New mnemonics may be used for screen control, e.g., begin blink, end blink, begin underline, end underline, normal video, reverse video; or for terminal messages, such as those illustrated below.

Refer to Utility *NPSD to define configurations.

'RC'	Prints the message "RETURN TO CONTINUE".
'YN'	Prints the message "? (Y/N)".

OUTPUT FORMATTING (MASKING)

Output formatting is used in printing numeric data to conform to certain formats used in commercial or financial applications. It is also used to right-justify or decimal-align output. This is done by specifying a string mask through which the numeric output will be printed. Output formatting can be used in outputting I/O directives, the STR function, and the IOLIST directive.

Format

numeric-variable: (mask)

The numeric variable is printed through the mask which is a string constant, variable or expression.

Special characters used in the mask to represent special functions:

- | | |
|--------|---|
| 0 | Places a numeric digit or 0 in the specified character position. |
| # | Replaces any insignificant 0 (leading or trailing) by a space in the specified character position. |
| * | Replaces a leading 0 with an asterisk in the specified character position. |
| \$ | Places a dollar sign in the character position immediately to the left of the most significant digit or the decimal point if there are no digits to the left of the decimal point. |
| , | Inserts a comma in the specified character position if the character to its left is a numeric digit. |
| . | Inserts a decimal point in the specified character position unless the rest of the string is spaces, in which case a space is inserted. |
| (mask) | Enclosing the mask string in parentheses causes the output string to be enclosed in parentheses if the numeric value is negative. If the output is zero or positive, spaces are inserted in place of parentheses. |
| + | Inserts a plus (+) or minus (−) sign at specified character position depending upon the value of the number; this may appear to the left or right of the value. |
| − | Inserts a minus (−) sign at the specified character position if the value is negative and a space if the value is zero or positive; this may appear to the left or right of the value. |
-

DR Inserts a debit (DR) at the specified character positions if the value is positive or zero and a credit (CR) if the value is negative; this specification must appear at the end of the mask.

CR Prints a credit (CR) at the specified character position if the value is negative, and spaces if the value is zero or positive; this specification must appear at the end of the mask.

character Any other character string than those listed above will be printed in the specified character position.

If the number to be printed with the mask has more digits to the left of the decimal point than the mask allows, the mask is ignored and the value is printed without formatting. If there are more digits to the right of the decimal point than the mask allows, the value is rounded to fit the mask.

Examples

(numeric value = 54321)

(numeric value = - 54321)

MASK	RESULT	MASK	RESULT
"###,###,###"	54,321	"##.00"	54321
"\$##,###,###"	\$54,321	"- ###,###"	- 54,321
"\$##,###,###.00"	\$54,321.00	"(###,###)"	(54,321)
"DM###,###.00"	DM 54,321.00	"###,### -"	54,321 -
"\$* #,###,###.00"	*****\$54,321.00	" + ###,###"	- 54,321

Usage

PRINT N:"####.00" if N = 54.1 prints "54.10"
 PRINT N:Q\$ if N = 54.1 and Q\$ = "\$###.00" prints "\$54.10"
 STR (N:T\$) if N = 54.1 and T\$ = "\$###.0" returns "\$54.1"

SPECIAL CHARACTER AND CODE REPRESENTATION

Thoroughbred/OS provides for the representation of certain special characters with the use of shorthand mnemonic constants. These constants may be used as an alternative to the hexadecimal representation of these codes that is sometimes necessary in order to prevent errors when using these characters.

Mnemonic Constants

- QUO** Represents the quote character ("). This is used to place a quote within a string without falsely representing the end of the string value.
- SEP** Represents the field separator character; also specifies a line feed and carriage return.
- ESC** Represents the escape character. The effect is device dependent.

Examples

`PRINT QUO + "ABC" + QUO` prints "ABC" (in quotation marks)

`WRITE (1) A$ = SEP = T` writes the file record in A\$ into the first field and T into the second field

`A$ = ESC + "CS"` if A\$ is printed has the same effect as when `PRINT 'CS'` allows mnemonics to be imbedded in a string

FUNCTION

This directive is used to transfer data from a file to specified variables in a program. It also places a restriction on the extracted record so that no other Task may access it. After execution, the record pointer values are not changed and still indicate the extracted record.

SYNTAX

EXTRACT (channel [,I/O-options]) [variable-list]
[,IOL = stmt-no]

EXTRACT RECORD (channel [,I/O-options])
string-variable

where:

channel is an integer value indicating the channel of an OPEN file

I/O-options is one or more of the following specifiers:

Record

IND = numeric-value

KEY = string-value

Branching

ERR = stmt-no

DOM = stmt-no

END = stmt-no

Misc

TBL = stmt-no

variable-list is a list of numeric or string variables that receive values from the file record

stmt-no (IOL) specifies a statement number containing an IOLIST that defines a variable list

string-variable is a single string variable that receives the entire record as data.

DESCRIPTION

- Access to an extracted record is restricted to the issuing Task until another I/O operation is conducted on the same file through the same channel by that Task.

- I/O options include:

IND = specifies the Index number of the record to access

KEY = specifies the Key value of the record to access

ERR = specifies the number of the statement to branch to if the directive produces an error condition

DOM = specifies the number of the statement to branch to if an attempt is made to access a record with a duplicate or missing Key value (error 11)

END = specifies the number of the statement to branch to if the end of the file is reached (error 2)

TBL = specifies the statement number location of a TABLE statement to be used for code conversion

The IND and/or END options cannot appear with the KEY and/or DOM options; they are mutually exclusive in the same directive.

- Records are accessed in sequential order by Key value for Direct and Sort files or Index value for Indexed files unless the KEY or IND options are used.
 - Values from the fields of the accessed record are loaded into the variable list or IOLIST in sequential order (i.e., the value of
-

the first field in the record is loaded into the first variable, the second value into the second variable, etc.). An asterisk (*) is used to specify a field that is skipped and will not have data entered into a variable (not used for EXTRACT RECORD).

- The RECORD modifier for this directive allows the entire record, including delimiting characters, to be entered as data into a single string variable. This modifier cannot be used with the IOL= option.
- An EXTRACT of a SORT file only advances the record pointer to the Key value of the extracted record.
- A WRITE operation that follows the EXTRACT (and is intended to write over the extracted record) does not need to have the Index or Key values specified since the record pointer points to the extracted record.
- This directive should not be used to input data from a terminal.

EXAMPLES

EXTRACT (1)A\$,A

accesses the current record in the file OPEN on channel 1 and transfers data from the first field to the variable A\$ and from the second field to the variable A. Access to the record is restricted; the record pointer is retained to indicate the extracted record

EXTRACT RECORD (1)B\$

accesses the current record in the file OPEN on channel 1 and transfers the entire record, including delimiting characters, into the variable B\$

EXTRACT RECORD (1,IND = X, ERR = 7999)B\$
if X=56; has the same effect as example two, but accesses the record having the Index number 56; branches to statement 7999 if the directive produces an error condition

EXTRACT (1, KEY = I\$, END = 7500)IOL = 5000
if I\$ = "ASD#123" and statement 5000 is IOLIST A\$,A, this has the same effect as the first example, but accesses the record with the Key value "ASD#123"; branches to statement 7500 if the end of the file is reached

EXTRACT (1,TBL = 459)*,A
has the same effect as the first example, but skips the first field and enters data from the second field into the variable A; accesses statement 459 for the TABLE statement to use for code conversion of the data

FUNCTION This directive is used to transfer data from a file to specified variables in a program. This directive will only advance the record pointer to the next sequential record if the FIND is successfully executed; otherwise, the record pointer will remain indicating the last record accessed before the unsuccessful execution of the FIND.

SYNTAX FIND (channel [,I/O-options]) [variable-list]
[,IOL = stmt-no]
FIND RECORD (channel [,I/O-options])
string-variable

where:

channel is an integer value indicating the channel of an OPEN file

I/O-options is one or more of the following specifiers:

Record

IND = numeric-value

KEY = string-value

Branching

ERR = stmt-no

DOM = stmt-no

END = stmt-no

Misc

TBL = stmt-no

variable-list is a list of numeric or string variables that receive values from the file record

stmt-no (IOL) specifies a statement number containing an IOLIST that defines a variable list

string-variable is a single string variable that receives as data the entire record

DESCRIPTION

- I/O options include:

IND = specifies the Index number of the record to access

KEY = specifies the Key value of the record to access

ERR = specifies the number of the statement to branch to if the directive produces an error condition

DOM = specifies the number of the statement to branch to if an attempt is made to access a record with a duplicate or missing Key value (error 11)

END = specifies the number of the statement to branch to if the end of the file is reached (error 2)

TBL = specifies the statement number location of a TABLE statement to be used for code conversion

The IND and/or END options cannot appear with the KEY and/or DOM options; they are mutually exclusive in the same directive.

- Records are accessed in sequential order by Key value for Direct and Sort files or Index value for Indexed files unless the KEY or IND options are used.
- If the FIND directive is successful, the record pointer is updated to point to the next sequential record. If the access is unsuccessful, the record pointer is not advanced and remains set to its previous values.

- Values from the fields of the record being accessed are loaded into the variable list or IOLIST in sequential order (i.e., the value of the first field in the record is loaded into the first variable, the second value into the second variable, etc.). An asterisk (*) is used to specify a field that is skipped and will not have data entered into a variable (not used for FIND RECORD).
- The RECORD modifier for this directive allows the entire record, including delimiting characters, to be entered as data into a single string variable. This modifier cannot be used with the IOL = option.
- A FIND on a SORT file does not transfer data since there is no data to transfer; the record pointer is updated if the FIND is successful.
- This directive should not be used to input data from a terminal.

EXAMPLES

FIND (1)A\$,A

accesses the current record in the file OPEN on channel 1 and transfers data from the first field to the variable A\$ and from the second field to the variable A. The record pointer is updated to the next sequential record if the FIND is successful; otherwise, an error is produced and the record pointer remains at the value of the previous record

FIND RECORD (1)B\$

accesses the current record in the file OPEN on channel 1 and transfers the entire record, including delimiting characters, into the variable B\$

FIND RECORD (1,IND = X, ERR = 7999)B\$

if X = 56; has the same effect as example two, but accesses the record having the

Index number 56; branches to statement 7999 if the directive produces an error condition

FIND (1, KEY = I\$, END = 7500)IOL = 5000

if I\$ = "ASD#123" and statement 5000 is IOLIST A\$,A, this has the same effect as the first example, but accesses the record with the Key value "ASD#123" and branches to statement 7500 if the end of the file is reached

FIND (1,TBL = 459)*,A

has the same effect as the first example, but skips the first field and enters data from the second field into the variable A; accesses statement 459 for the TABLE statement to use for code conversion of the data

FUNCTION

This directive reads data from a disk by accessing a specific disk location rather than a file.

SYNTAX

GET disk-no, sector-no [,ERR = stmt-no],
string-variable

where:

disk-no is an integer value specifying
the logical disk to access

sector-no is an integer value specifying
the number of the sector to
start reading from

stmt-no is the number of the statement
to branch to if this directive
produces an error condition

string-variable is a string variable that will
receive the data to be read
from the disk

DESCRIPTION

- The string variable used to receive the data read from the disk must consist of only a single letter and must be already dimensioned with the DIM directive.

EXAMPLES

GET 2,4,A\$

accesses disk 2 and reads data into the pre-dimensioned variable A\$

GET X,Y, ERR = 7800,A\$

if X = 2 and Y = 4, has the same effect as the first example and will branch to statement 7800 if an error occurs while processing this directive

FUNCTION

This directive is used to transfer data to specified variables in a program from either a terminal or a file. If a terminal is specified, a variety of options are included to allow for the output and positioning of data messages, the positioning of the cursor, the execution of special terminal routines and the verification of INPUT data. If a file is specified, this directive acts in a way similar to the READ directive.

SYNTAX

INPUT (channel [,I/O-options])
 [@(column[,row])] [,mnemonic] [,output]
 [,variable-list] [:verification] [IOL = stmt-no]
INPUT RECORD (channel [,I/O-options])
 string-variable

where:

channel is an integer value indicating
 the channel of an OPEN file

I/O-options is one or more of the following
 specifiers:

Branching

ERR = stmt-no

DOM = stmt-no

END = stmt-no

Misc

TBL = stmt-no

SIZ = numeric-value

TIM = numeric-value

LEN = numeric-value

Record

IND = numeric-value

KEY = string-value

*column[,row] is one or two numeric values
 that specify the column-row
 positions at which to display
 output or position the cursor

- *mnemonic is a two character code that indicates a special procedure to be performed on the terminal or device
- *output is a string constant output to the terminal or device, usually a prompting message
- *variable-list is a list of numeric or string variables that receive values from the file record
- *verification is an Input Verification modifier that specifies certain parameters necessary for values that are input
- *stmt-no (IOL) specifies a statement number containing an IOLIST that defines a variable list

string-variable is a single string variable that receives as data an entire sequence of terminal responses or an entire record

*indicates that these syntax components may appear in other logical sequences than the one shown here

DESCRIPTION

- I/O options include:

ERR = specifies the number of statement to branch to if the directive produces an error condition

DOM = specifies the number of the statement to branch to if an attempt is made to access a record with a duplicate or missing Key value (error 11)

- END = specifies the number of the statement to branch to if the end of the file is reached (error 2)
- TBL = specifies the statement number location of a Table statement to be used for code conversion
- SIZ = the size of data to be INPUT before a field delimiter is automatically returned
- LEN = the maximum number of characters of data that will be allowed to be entered
- TIM = indicates the number of seconds allowed to elapse without any INPUT before an ERROR 0 is returned
- IND = specifies the INDEX number of the record to access
- KEY = specifies the KEY value of the record to access

The IND and/or END options may not appear with the KEY and/or DOM options; they are mutually exclusive in the same directive. TIM is mutually exclusive of KEY or IND.

- Refer to Data Positioning, Mnemonics, and Input Verification for a detailed discussion of these options.
- Values input from a terminal or file are loaded into the variable list or IOLIST in sequential order. The first data value entered from a terminal or the first field in a record is loaded into the first variable, the second into the second variable, etc. An

asterisk (*) is used to specify an entry or field that is skipped and will not have data entered into a variable (not used for INPUT RECORD).

- The RECORD modifier is used when specifying a file to allow an entire record, including delimiting characters, to be entered as data into a single string variable. This modifier cannot be used with the IOL=, data positioning, mnemonic, or verification options.
- If a file is specified as the input source, the records are accessed in sequential order by Key value for Sort or Direct files and Index value for Indexed files unless the IND or KEY options are used. After the directive is executed, the record pointer is advanced to indicate the next sequential record.
- For an INPUT directive directed to a terminal, each value entered is followed by a Carriage Return (or a Function or Control Key) to end the entry. Or, you may use the SIZ= specifier that automatically ends the entry after the specified number of characters is entered.
- This directive also sets the value of the CTL Task variable depending upon which Control or Function Key was depressed.

EXAMPLES

INPUT A\$,B

provides for the input of data from the User terminal (an unspecified channel number defaults to 0, the User Task VDT) into the variables A\$ and B; the entry of each value is separated by a Carriage Return or Control or Function Key. If the data for the first variable is not a string value, or for the second not a numeric value, an error is produced

INPUT *

allows for temporary program suspension until a Carriage Return or Function or Control Key is depressed; the CTL variable is set according to which Key is returned; no variable is assigned a value by this form of the directive

INPUT (0,SIZE = 4) @ (4,10),“ENTER ID:”,A\$,B\$
prints, starting at column 4, row 10 of the terminal, the message “ENTER ID:” and positions the cursor after this message to accept entry of data; the first 4 characters are entered into the variable A\$ and the remaining characters into B\$

INPUT (0,ERR = 7999)‘CS’ @ (2),“FROM”,X:(123)
clears the screen (‘CS’) and prints the message “FROM”, starting at column 2 and accepts a numeric value from 0 to 123; if the value entered is out of this range, an error is produced and execution branches to statement 7999

INPUT RECORD (1,IND = X,END = 5000)A\$
if X = 24, accesses the file OPEN on channel 1 and inputs the record having the Index value 24; the entire record including delimiting characters is loaded into the variable A\$. Execution branches to statement 5000 if an attempt is made to access a record beyond the end of the file

FUNCTION

This statement is used to form an input/output list that may be used for most I/O directives. By defining this list, it may be used to refer to I/O directives throughout the program without needing to define the list in each directive.

SYNTAX

IOLIST [**@**(column[,row])] [,mnemonic]
[,output] [,variable-list] [:verification]
[:masking] [,IOL = stmt-no]

where:

- *column[,row] is one or two numeric values that specify the column-row positions at which to display output or position the cursor
- *mnemonic is a two character code that indicates a special procedure to be performed on the terminal or device
- *output is a string constant output to the terminal or device, usually a prompting message
- *variable-list is a list of numeric or string variables that receive or supply values for the I/O directive
- *verification is an input verification modifier that specifies certain parameters necessary for values that are input
- *masking is a string value used as the mask for formatting the output of numeric variables

*stmt-no (IOL) specifies a statement number containing an IOLIST that defines a variable list

*indicates that these syntax components may appear in logical sequences other than the one shown here.

DESCRIPTION

- The IOLIST may be used to define variable lists, input verification, mnemonic device control, string text output, data positioning and other IOLISTS.
- The IOLIST is referred to within the I/O directives by using the IOL= option to specify the statement number at which the desired IOLIST is located.

- Restrictions

Data positioning may not be used with the READ, EXTRACT and FIND directives.

Verification can be used only with the INPUT directive.

IOLISTS may not be used with RECORD modified directive.

Masking may be used for the output directives PRINT and WRITE only.

- Refer to the sections on Input Verification, Data Positioning, Output Formatting and Device Control for a more detailed description of these options.

EXAMPLES

IOLIST X1,Y2\$,Z3

defines a variable list for the input or output of data that will be appended to the I/O directive

IOLIST X1,IOL = 200

defines an IOLIST with the variable X1 and the contents of the IOLIST located at statement 200

IOLIST @(2,4), "ENTER ID", 'RB', X\$:(LEN = 2,4)

defines an IOLIST for an INPUT directive that includes data positioning, string output, and input verification

FUNCTION

This directive is used to output data from the specified variables to a terminal, printer or a file. If a terminal is specified, a variety of options are included to allow for the output and positioning of data messages, the positioning of the cursor and the execution of special terminal and printer procedures. If a file is specified, this directive acts similarly to the WRITE directive.

SYNTAX

```
PRINT (channel [,I/O-options])  
    [@(column[,row])] [,mnemonic] [,output]  
    [,variable-list] [:masking] [,IOL = stmt-no]  
PRINT RECORD (channel [,I/O-options])  
    string-variable
```

where:

channel is an integer value indicating
 the channel of an OPEN file

I/O-options is one or more of the following
 specifiers:

Branching

ERR = stmt-no

DOM = stmt-no

END = stmt-no

Misc

TBL = stmt-no

Record

IND = numeric-value

KEY = string-value

*column[,row] is one or two numeric values
 that specify the column-row
 positions at which output is
 printed or where the cursor is
 positioned

- *mnemonic is a two character code that indicates a special procedure to be performed on the terminal or device specified
- *output is a string constant output to the terminal or device
- *variable-list is a list of numeric or string variables that contain values to be output
- *masking is a string value that is used as the mask in outputting numeric variables
- *stmt-no (IOL) specifies a statement number containing an IOLIST that defines a variable list

string-variable is a single string variable that outputs an entire record

*indicates that these syntax components may appear in logical sequences other than the one shown here

DESCRIPTION

- I/O options include:

ERR = specifies the number of the statement to branch to if the directive produces an error condition

DOM = specifies the number of the statement to branch to if an attempt is made to access a record with a duplicate or missing Key value (error 11)

END = specifies the number of the statement to branch to if the end of the file is reached (error 2)

TBL = specifies the statement number
location of a TABLE statement to
be used for code conversion

IND = specifies the INDEX number of the
record to access

KEY = specifies the KEY value of the
record to access

The IND and/or END options cannot appear
with the KEY and/or DOM options; they are
mutually exclusive in the same directive.

- Refer to Data Positioning, Output Formatting (masking) and Mnemonics for a detailed description of these options.
- Values printed to a terminal or file are output from the variable list or IOLIST in sequential order. The first data variable output is printed first to either the output device or to the first field in a record.
- The directive will output all of the device information on the right of the channel specifier to a file. This allows the data positioning and mnemonics to be printed as data to the file and allows it to be accessed later with the RECORD modifier and the value printed out. The result is the same as if the original PRINT statement were executed directly.
- The RECORD modifier is used to allow an entire record, including delimiting characters, to be output as data from a single variable. This modifier cannot be used with the IOL=, data positioning, mnemonic, or verification options, and is usually directed to a file.

- If a file is specified as the output form, the records are accessed in sequential order by Key value for Direct and Sort files or Index value for Indexed files unless the IND or KEY options are used. After this directive is executed, the record pointer is advanced to indicate the next sequential record.

EXAMPLES

PRINT A\$,B

prints at the terminal (an unspecified channel defaults to 0, the User Task VDT) the values in variables A\$ and B

PRINT A\$,B,

has the same effect as the first example, but the last comma suppresses the automatic line feed after the last value is printed

PRINT @(4,10),"ENTER ID:",A\$

prints, starting at column 4, row 10 of the terminal, the message "ENTER ID:" and then the variable A\$

PRINT (0,ERR = 7999)'CS',@(2),"FROM",X

clears the screen ('CS'), prints the message "FROM" starting at column 2, prints the numeric value X and branches to statement 7999 if an error is produced

PRINTRECORD (2,IND = X)A\$

if X = 56, prints to the file OPEN on channel 2 at the record with the Index number 56 the contents of A\$

PRINT (2,END = 8500)@(4,10),"ENTER D ID:",A\$

prints to the current record of the file OPEN on channel 2 all of the data to the right of ...END = 8500), a subsequent READ RECORD of this same record and a PRINT or WRITE of the value obtained will produce the same

result as if the PRINT directive in the third example had been executed directly. If an attempt is made to PRINT past the end of the file, execution branches to statement 8500

FUNCTION

This directive writes data to the disk by accessing a specific disk location rather than a file.

SYNTAX

PUT disk-no, sector-no [,ERR = stmt-no],
string-variable [:verification]

where:

disk-no	is an integer value specifying the logical disk to access
sector-no	is an integer value specifying the number of the sector to start writing to
stmt-no	is the number of the statement to branch to if this directive produces an error condition
string-variable	is a string variable that holds the data to be written to the disk
verification	is a string variable that contains a value used as a match comparison with the data to be written

DESCRIPTION

- The string variable used to hold the data to be written to the file must consist of only a single letter. The verification variable must also be a single letter.
- For the verification option, if the data to be written and the data in the verification variable do not match, an error is returned.
- Note: Care must be used when executing this directive since it will have the effect of irretrievably writing over data on the disk.

- This directive can only be used on a disabled disk.

EXAMPLES

PUT 2,4,A\$

accesses disk 2 and writes the data contained in A\$ onto the disk starting at sector 4

PUT X,Y,ERR=7800,A\$,B\$

if $X = 2$ and $Y = 4$ has the same effect as the first example and will branch to statement 7800 if an error occurs while processing this directive; if the values of A\$ and B\$ are not matched, an error is produced

FUNCTION This directive is used to transfer data from a file to specified variables in a program. This directive also advances the record pointer to indicate the next sequential record.

SYNTAX READ (channel [,I/O-options]) [variable-list]
[IOL = stmt-no]
READ RECORD (channel [,I/O-options])
string-variable

where:

channel is an integer value indicating the channel of an OPEN file

I/O-options is one or more of the following specifiers:

Record

IND = numeric-value

KEY = string-value

Branching

ERR = stmt-no

DOM = stmt-no

END = stmt-no

Misc

TBL = stmt-no

variable-list is a list of numeric or string variables that receive values from the file record

stmt-no (IOL) specifies a statement number containing an IOLIST that defines a variable list

string-variable is a single string variable that receives as data the entire file record

DESCRIPTION

- I/O options include:

IND = specifies the Index number of the record to access

KEY = specifies the Key value of the record to access

ERR = specifies the number of the statement to branch to if the directive produces an error condition

DOM = specifies the number of the statement to branch to if an attempt is made to access a record with a duplicate or missing Key value (error 11)

END = specifies the number of the statement to branch to if the end of the file is reached (error 2)

TBL = specifies statement number location of a TABLE statement to be used for code conversion

The IND and/or END options cannot appear with the KEY and/or DOM options; they are mutually exclusive in the same directive.

- Records are accessed in sequential order by Key for Direct and Sort files or by Index value for Indexed files unless the KEY or IND options are used.
- Values from the fields of the record being accessed are loaded into the variable list or IOLIST in sequential order (i.e., the value of the first field in the record is loaded into the second variable, etc.). An asterisk (*) is used to specify a field that is skipped and will not have data entered into a variable (not used for READ RECORD).

- The RECORD modifier for this directive allows the entire record, including delimiting characters, to be entered as data into a single string variable. This modifier cannot be used with the IOL = option.
- A READ of a SORT file does not transfer data; it advances the record pointer to the next sequential record (next Key value).
- This directive should not be used to input data from a terminal.

EXAMPLES

READ (1)A\$,A

accesses the current record in the file OPEN on channel 1 and transfers data from the first field to the variable A\$ and from the second to the variable A; the record pointer is updated to the next record

READ RECORD (1)B\$

has the same effect as the first example, but transfers the entire record, including delimiting characters, into the variable B\$

READ RECORD (1,IND = X,ERR = 7999)B\$

if X = 56, has the same effect as example two but accesses the record having the Index number 56, and branches to statement 7999 if the directive produces an error condition

READ (1,KEY = I\$,END = 7500)IOL = 5000

if I\$ = "ASD#123" and statement 5000 is IOLIST A\$,A, this has the same effect as the first example, but accesses the record with the Key value "ASD#123"; branches to statement 7500 if the end of the file is reached

READ (1,TBL = 459)*,A

has the same effect as the first example, but skips the first field and enters data from the

second field into the variable A; accesses statement 459 for the TABLE statement to use for code conversion of the data

REMOVE Directive

REMOVE

FUNCTION

This directive is used to remove a Key and record from a Direct or Sort file by deleting the Key specified and writing over the data area with null characters (\$00\$). The record pointer is advanced to indicate the next sequential Key value record.

SYNTAX

REMOVE (channel,KEY = string-value
[,I/O-options])

where:

channel is a numeric value indicating the channel of an OPEN file

string-value is a string value that represents a Key value for the record

I/O-options is one or more of the following branching specifiers:
DOM = stmt-no
ERR = stmt-no

DESCRIPTION

- This directive cannot be applied to other file types or an error is produced.
- The I/O options for this directive are:
 - DOM = specifies the number of the statement to branch to if this directive produces a duplicate or missing Key error (error 11)
 - ERR = specifies the number of the statement to branch to if this directive produces an error condition

EXAMPLES

REMOVE (5,KEY = "A245")
deletes the record from the file that is OPEN on channel 5 that is indicated by the Key value "A245"

REMOVE (5,KEY = K\$,DOM = 5000,ERR = 7999)
if K\$ = "A245", has the same effect as the
first example and will branch to statement
5000 if a duplicate or missing Key condition
exists or to statement 7999 if any other error
condition occurs

FUNCTION

This statement defines a conversion table that is used to convert data, either data input with an input directive, or data to be output with an output directive, into a different form.

SYNTAX

TABLE mask table

where:

mask is a single byte, expressed as two hexadecimal digits, that is used as one byte in the logical AND operation with the input byte

table is a sequence of bytes expressed as two hexadecimal digits that are used to specify the output for each input byte

DESCRIPTION

- There are a maximum of 256 data codes for possible conversion. A complete **TABLE** will, therefore, require 256 bytes (512 hexadecimal digits) for conversion specifications, but for many cases it will be only necessary to define a portion of the table since many of the possible input codes are not used or can be filtered out.
- The filtering of data to be converted is accomplished by the use of a mask which is used as one byte of the logical AND operation, with the other byte being the data to convert. If bit *n* of the mask is 1, the corresponding bit of the data byte is transmitted unchanged to the table. If it is 0, a 0 is transmitted to the table.
- As each data byte is operated on by the logical AND with the mask byte, the resulting byte is treated as a number to determine which byte of the table will be used to sup-

ply the translated code. For example, if the data byte is \$D7\$ ("W" or binary 11010111) and the mask byte is \$7F\$ (binary 01111111) then the result is \$57\$ (binary 01010111) which is equal to the decimal value 87. The 87th byte of the table is, therefore, used as the character to represent the original data byte.

- The TABLE definition is referred to in the I/O directives by the TBL= option that specifies the statement number of the TABLE statement to be used in the conversion.
- When used with input directives, the table is applied before the record is scanned for delimiting characters. Thus, the delimiting or control characters, as well as the data, will be translated before the system interprets the record. When used with the output directives, the translation is applied after the system adds the delimiting of control characters to the record.
- When applied to records of SORT or DIRECT files, the Key values of the records are also translated.

EXAMPLES

TABLE FF	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
	20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F
	30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F
	40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F
	50	51	52	53	54	55	56	57	58	59	5A	5B	5C	5D	5E	5F
	60	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F
	50	51	52	53	54	55	56	57	58	59	5A	7B	7C	7D	7E	7F
	80	81	82	83	84	85	86	87	88	89	8A	8B	8C	8D	8E	8F
	90	91	92	93	94	95	96	97	98	99	9A	9B	9C	9D	9E	9F
	A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	AA	AB	AC	AD	AE	AF
	B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	BA	BB	BC	BD	BE	BF
	C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	CA	CB	CC	CD	CE	CF
	D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	DA	DB	DC	DE	DE	DF
	E0	E1	E2	E3	E4	E5	E6	E7	C8	C9	CA	CB	CC	CD	CE	CF
	D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	DA	FB	FC	FD	FE	FF

(Spaces are shown for clarity only and should not be included when entering the data.)

Defines a TABLE for translating ASCII code with mixed upper and lower case letters to all upper case letters. All characters other than the 26 lower case letters are unchanged (see Appendix A). This table is designed to work regardless whether bit 8 is set to "0" or "1" by the system.

FUNCTION

This directive is used to transfer data from the specified variables in a program to a file or device. The record pointer is advanced to the next sequential record.

SYNTAX

```
WRITE (channel [,I/O-options]) [variable-list]
      [:masking] [,IOL = stmt-no]
WRITE RECORD (channel [,I/O-options])
      string-variable
```

where:

channel is an integer value indicating the channel of an OPEN file

I/O-options is one or more of the following specifiers:

Record

IND = numeric-value

KEY = string-value

Branching

ERR = stmt-no

DOM = stmt-no

END = stmt-no

Misc

TBL = stmt-no

variable-list is a list of numeric or string variables whose values are transferred to the file or device

masking is a string value used as the mask in outputting numeric values

stmt-no (IOL) specifies a statement number containing an IOLIST that defines a variable list

string-variable is a single string whose value is transferred to the file or device

DESCRIPTION

- I/O options include:

IND = specifies the Index number of the record to access

KEY = specifies the Key value of the record to access

ERR = specifies the number of the statement to branch to if the directive produces an error condition

DOM = specifies the number of the statement to branch to if an attempt is made to access a record with a duplicate or missing Key value (error 11)

END = specifies the number of the statement to branch to if the end of the file is reached (error 2)

TBL = specifies statement number location of a TABLE statement to be used for code conversion

The IND and/or END options cannot appear with the KEY and/or DOM options; they are mutually exclusive in the same directive.

- Records are accessed in sequential order by Key value for Direct and Sort files or Index value for Indexed files unless the KEY or IND options are used.
- Values of the variables are transferred from the variable list or IOLIST to the fields of a record in sequential order (i.e., the value of the first variable is loaded into the first field of the record, the second value into the second record, etc.).

- The RECORD modifier for this directive allows the entire record, including delimiting characters, to be entered as data from a single string variable. This modifier cannot be used with the IOL = option.
- A WRITE to a Sort file only establishes the Key value of the record and advances the pointer to the next sequential record.
- If directed to a terminal, the output is printed starting at the current cursor position and a carriage return is printed after each variable. Data positioning is not included as an option for this directive.
- If a WRITE is directed to a Direct file, without specifying the Key value, each record to be written to must be extracted first. The Key value will not be updated for a record that has been extracted until it has been written to.

EXAMPLES

WRITE (1)A\$,A

accesses the current record in the file OPEN on channel 1 and transfers data from the first variable A\$ into the first field of the record and from the variable A into the second field; the record pointer is updated to the next sequential record

WRITE RECORD (1)B\$

has the same effect as the first example, but transfers all of the data in the variable, including delimiting characters, into the current record of the file

WRITE RECORD (1,IND = X,ERR = 7999)B\$

if X = 56, has the same effect as example two, but accesses the record having the Index number 56 and branches to statement 7999 if the directive produces an error condition

WRITE (1,KEY = I\$,END = 7500)IOL = 5000

if I\$ = "ASD#123" and statement 5000 is
IOLIST A\$,A, this has the same effect as the
first example, but accesses the record with
the Key value "ASD#123" and branches to
statement 7500 if the end of the file is
reached



CHAPTER SEVEN—PUBLIC PROGRAMS

In a typical multi-user environment, where more than one user may be using the same program at one time, a separate copy of this program is kept in RAM memory for each user. By establishing this "common" program as a Public Program, a single program may be kept in RAM memory that can be shared by users operating in the same memory bank, greatly decreasing program memory requirements. Each user may then access this single, shared copy and use it to process a unique problem.

Establishing a Public Program also allows it to serve as a "sub-routine" program. You may "call" a Public Program from another program and send values to it. The Public Program is then executed. When the Public Program is finished processing, you are returned to the point where you left your original program. At this point you may also pass values from the Public Program back to your original program.

When a program is established as a Public Program, an entry in the File Control Table is made for the program that records disk location parameters. Because references to a Public Program can refer to this table, there is no need for time consuming disc directory searches. This allows for faster accessing and execution time.

PUBLIC PROGRAM PROCESS

Public Programming involves the following series of steps:

1. File Control Table Entry

An entry is made in the File Control Table that records program location information.

2. Load Program into RAM Memory

A single copy of the program is loaded into RAM memory and designated as a Public Program.

3. Execution of the Program

The Public Program is executed and values may be passed to it by the "calling" program. When finished, the "calling" program resumes execution at the point where it called the Public Program and values may be passed back to the calling program.

4. Remove a Program from RAM Memory
A program is removed from RAM memory.
5. Remove a File Control Table Entry
The program entry in the File Control Table is removed.

Public Program Directives

The Public Program directives are:

ADDR
CALL
ENTER
EXIT

For additional directives and functions associated with the use of Public Programming, see the ADD, DROP, and END directives and the PUB function.

Restrictions on Public Programs

- A Public Program cannot execute the following directives: DELETE, EXECUTE, LIST, MERGE, RUN, SAVE
- A non-Public Program cannot execute the following directives: ENTER, EXIT

Transfer of Values

Values may be transferred between a "calling" and a "called" Public program by utilizing the value list of the CALL directive to send values to the Public program and the variable list of the ENTER directive to receive these values within the Public program. The values and variables in both lists must be of corresponding type and there must be an equal number of each. In order to send or receive an entire dimensioned numeric array, the word "all" is inserted into the subscript position to indicate that every element of the array is to be transferred (i.e., A(ALL) transfers or receives an entire array).

Whether or not a value is returned to the calling program in a changed form is determined by the form of the values and variables

in the CALL and ENTER lists. When a Public program is completed, the variables in the CALL value list may now have new values determined in the Public program. The following scheme shows how the change of values is determined:

CALL value	ENTER variable	Is the CALL value changed to equal the new ENTER variable value after the Public Program is finished executing?
D	D	yes
D	C	yes
D + (expression)	D	no
D(1)	E	no
D(ALL)	E(ALL)	yes
24	E	no
D\$	E\$	yes
"ABC"	E\$	no

FUNCTION This directive loads a program into memory and assigns it status as a resident Public Program.

SYNTAX ADDR program-name [,ERR = stmt-no]
 [,BNK = bank-no]

where:

program-name is a string value that specifies
 a program name

stmt-no is the number of the state-
 ment to branch to if an error
 occurs while processing this
 directive

bank-no is an integer value specifying
 the memory bank the program
 is loaded into

- DESCRIPTION**
- A program added via ADDR remains in the assigned memory bank until removed with the DROP directive.
 - The number of programs that can be loaded into a memory bank is limited by the capacity and status of entries already in the File Control Table.
 - Programs added to the memory bank with this directive will not displace any non-Public program in the user's Task area.
 - A program added via ADDR will be listed by the PUB function.
 - All configured memory banks may be accessed.

EXAMPLES

ADDR "INDEX"

loads the program "INDEX" into the user's current memory bank as a resident Public Program

ADDR A\$,ERR = 7999,

if A\$ = "INDEX", has the same effect as the previous example, but will branch to statement 7999 if an error occurs while processing this directive

FUNCTION

This directive executes the specified program as a Public Program and may be used to send values to and receive new values from the Public Program.

SYNTAX

CALL program-name [,ERR = stmt-no]
[value-list]

where:

program-name is a string value that specifies a program name

stmt-no is the number of the statement to branch to if an error occurs while processing this directive

value-list is a list of values (constants, variables or expressions) passed to the Public Program and which may receive new values returned from the Public Program

DESCRIPTION

- With this directive, a program that has not been assigned Public Program status with the ADDR directive will be loaded into memory as a Public Program and will be removed from the memory bank after execution of the program is completed.
- When a called program is completed, execution of the calling program is transferred to the statement following the CALL directive.
- This directive may be executed from a Public or a non-Public Program.

- The optional value-list contains values (constants, variables or expressions) that are used to send values to the variables in corresponding positions in the ENTER directive within the called program. The form of the values and variables in both lists determines whether a value is returned to the calling program (refer to the Introduction to this chapter).

EXAMPLES

CALL "INDEX"

loads and executes the program name "INDEX" as a Public Program

CALL A\$, ERR = 7999, A, D (ALL), C\$, "XYZ",
X\$ + "1"

if A\$ = "INDEX", has the same effect as the first example and will execute statement 7999 if an error occurs while processing this directive; a list of values that will be passed to and may receive returned values from the Public Program is also defined

FUNCTION This directive receives values passed to a Public Program from the CALL directive.

SYNTAX ENTER variable-list

- DESCRIPTION**
- Only one ENTER directive may appear in a program.
 - An ENTER directive cannot be executed in a program that is being RUN; it can only be executed in a program that is executing as a result of the CALL directive.
 - The variables in this list will receive values from the corresponding positions of the value list in the CALL directive. The values must be of corresponding data types, or an error will be produced.
 - This directive may appear at any point in a called program, but values are only received when the directive is executed.
 - Variables in this list may also be passed back to the calling program from the Public Program depending upon the form of the values in both lists (refer to the Introduction to this chapter).

EXAMPLES

CALL "INDEX", A\$, B, C (ALL)

ENTER M\$, M, O (ALL)

the Public Program "INDEX" will receive the value of A\$ in the variable M\$, the value of B in M and all of the elements of the dimensioned array C in corresponding elements of the array O

FUNCTION

This directive terminates a Public Program and allows the ERR variable in the calling program to be set to a specified value.

SYNTAX

EXIT [error-value].

where:

error-value is an integer value from 0 to 127 that is the value the ERR variable will be set to in the calling program

DESCRIPTION

- The execution of this directive transfers program control to the statement following the originating CALL directive.
- This directive may not be executed in a non-Public Program that is being processed by the RUN directive.
- A Public Program may also be terminated by an END directive, but the ERR variable will not be set in the calling program.

EXAMPLES**EXIT**

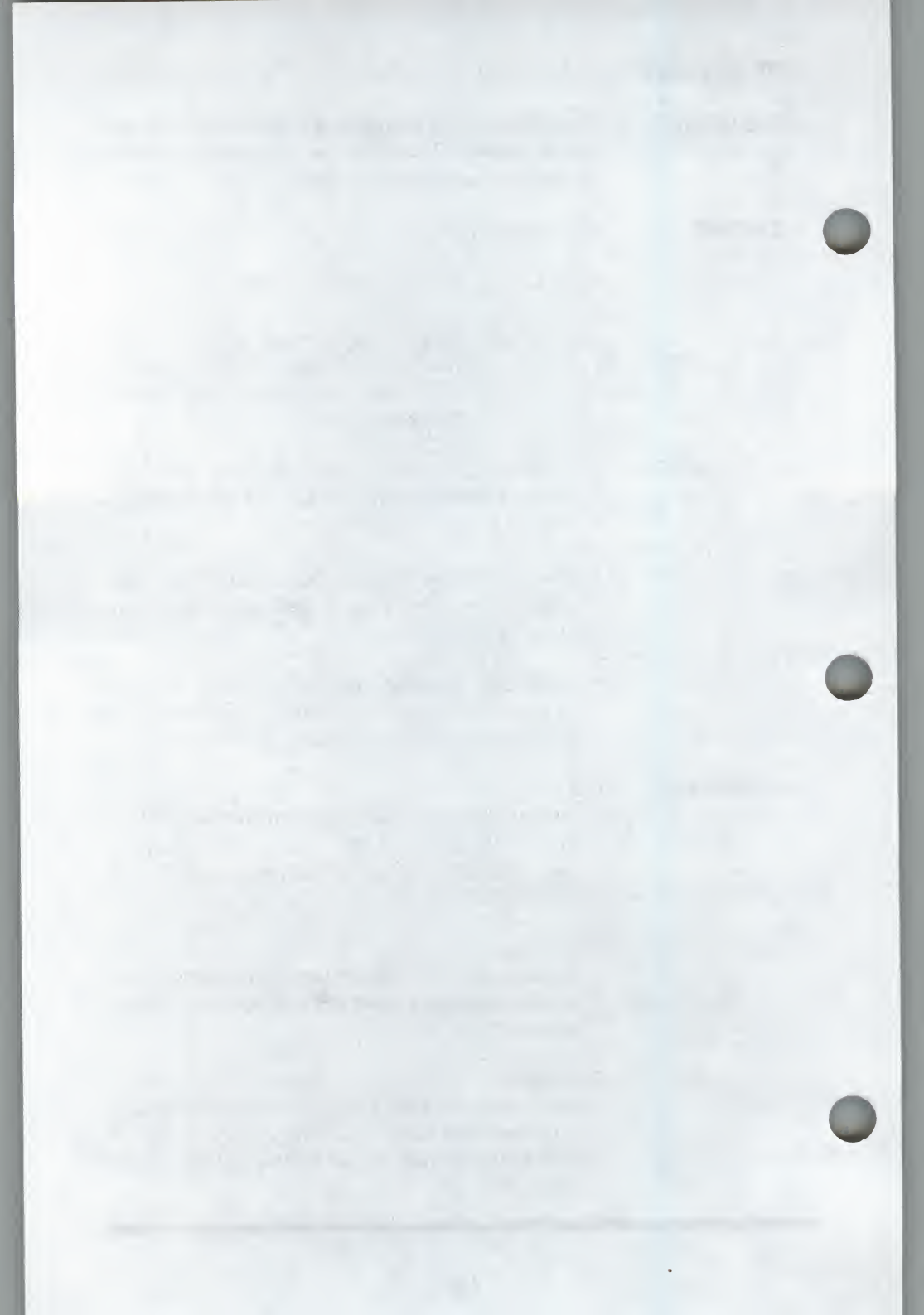
returns program execution from the "called" Public Program to the statement following the originating CALL directive; the ERR variable is not set

EXIT 12

has the same effect as the first example and in addition, sets the ERR variable in the calling program to 12

EXIT ERR

has the same effect as the first example and sets the ERR variable in the calling program to the value that it had in the called Public Program



CHAPTER EIGHT—ERROR PROCESSING

Thoroughbred/OS handles two types of errors as they occur: syntax errors and execution errors. This chapter details the procedures and capabilities for handling syntax and execution errors, and provides a detailed list of execution error codes, which includes information on the type of error that occurred.

SYNTAX ERRORS

Syntax errors are in the structure of a single statement, usually consisting of a spelling error or the improper construction of a statement. They are usually produced when statements are keyed in and the user is immediately notified by an error message in the following form:

```
*ERR          V
01000 LET X = = 10
```

The V indicates the location of the first character in the statement that violates a syntax rule as it is evaluated from left to right. Only one syntax error is returned at a time, even if more than one exists in the statement. Syntax errors may be corrected either by retyping the statement or by using the EDIT directive. If other syntax errors still exist in the statement, the statement is again returned with the position of the new error indicated. If no other errors are present, the statement is accepted.

EXECUTION ERRORS

Execution errors are produced by violations of logic or attempts to carry out procedures that cannot be processed due to resource conflicts or improper specifications. Execution errors are classed according to the error code list following these sections. Execution errors may be handled either by error processing and trapping routines, or by the suspension of processing and the return of the program to Console Mode operation. If an error causes a return to Console Mode, the program may be resumed, after correcting the error, with the RUN directive. The following sections detail the handling of execution errors.

ERROR TRAPPING AND PROCESSING CAPABILITIES

Thoroughbred/OS provides for certain error trapping and processing capabilities that handle the occurrence of an error within a program by directing program processing to a statement intended to correct or handle the error condition. These include the following:

- **END =** option directs branching to a specified statement if an end of file error (error 2) occurs.
- **DOM =** option directs branching to a specified statement if a duplicate or missing Key error (error 11) occurs.
- **ERR =** option directs branching to the specified statement if an unspecified error occurs.
- **SETERR** directive branches to a specified statement if an otherwise untrapped error occurs.

If more than one error trapping return is present, they are processed in the following sequence, starting with the first to be evaluated:

1. **END =** or **DOM =**
2. **ERR =**
3. **SETERR**

If none of these are present and an error occurs, execution is suspended and the error is displayed.

Untrapped Errors

For untrapped errors, the program will suspend execution, return to Console Mode, and print an error message in the following format:

```
!ERROR = 12  
09870 OPEN (1) "tty"
```

The error code number that specifies the type of error that occurred is returned, along with the statement in which the error occurred.

ERROR TASK VARIABLE

The ERR variable is a Task variable that returns the value of the last execution error to occur. This variable can be used to create error processing routines, or to construct conditional branches on specific error conditions. It can also be utilized in conjunction with the ERR function, which returns a value dependent upon the position of the error code in an Error-list, the IF/THEN/ELSE directive, or the ON GOTO or ON GOSUB directives to create selective error processing routines.

ERROR CODE LISTING

The following list gives the error code number of execution errors that may occur along with the error classification and its possible source.

Type/Code

Code: Meaning/Possible Causes

SYSTEM

- 1

DIRECTIVE OR FUNCTION NOT AVAILABLE

This release of Thoroughbred/OS or of your operating system does not support the directive or function requested.

INPUT/OUTPUT

00

FILE OR RECORD OR DEVICE BUSY/TIMEOUT ERROR

The program or task has attempted to:

- Access a terminal device that is not "Ready", (e.g., POWER-OFF or OFF-LINE).
- DISABLE a disk on which there is an OPEN file.
- DISABLE a disk which is already DISABLED.
- ERASE an OPEN file.
- Access a disk record which has been EXTRACTed by aother task.
- OPEN a file which has been LOCKed by another task.
- SAVE an ADDED program.
- SAVE to an OPEN or LOCKed file.

- Access a file which has been LOCKed by another task.
- LOCK a file which is OPEN to another task.
- WRITE to a Ghost Task which has not executed a corresponding INPUT.
- Communicate with a terminal device with the TIM = option, where the specified TIME has elapsed.
- START a task which has already been STARTed.

01

END-OF-RECORD

The program or task has attempted to:

- READ a record having a missing field terminator.
- READ more fields than the record contains.
- WRITE a record longer than the defined record length.
- Execute any I/O directive which specifies more variables than the field terminator characters received.
- PRINT more characters than the defined line length (for a printer terminal).
- WRITE beyond the end of the file OPENed with the ISZ = option when the last record to be written is larger than the available space remaining in the file.

02

END-OF-FILE

The program or task has attempted to:

- Execute an I/O directive to an INDEXED file with the IND = option specified greater than the defined file size.
- WRITE to a DIRECT or SORT file more records than the defined file size.
- Execute a sequential READ from a DIRECT or SORT file when the file pointer is at the highest value key.
- Reference the KEY or IND function when the file pointer is at the last record.
- READ or WRITE beyond the last record of a file OPENed with the ISZ = option.

- READ from a Ghost Task which is not in the output mode.
- MERGE an INDEXED file containing a program with no END statement.

04

DISK DRIVE SELECT

The program or task has attempted to:

- Define a disk file on a logical disk number greater than the maximum number physically configured for the system.
- Execute an I/O directive to a disk drive that is not "Ready".
- WRITE to a disk drive having the READ ONLY or PROTECT switch on.

06

INCOMPATIBLE DISK DIRECTORY

The program or task has attempted to reference a disk whose directory is configured incorrectly for the system.

07

SECTOR NUMBER/POINTER OUT OF RANGE

The program or task has attempted to:

- Reference disk sector 0 (for some operating systems) or reference a sector number higher than the highest available sector number.
- Execute an I/O directive to a DIRECT or SORT file which has out-of-range data pointers.

08

DISK WRITE VERIFICATION

The program or task has attempted unsuccessfully to use the Verification option with a GET or PUT directive.

FILE USAGE

10

FILE ID SIZE OR KEY USAGE

The program or task has attempted to:

- Specify a file-id containing either 0 or more than 8 characters.
- Execute an I/O directive to a DIRECT or SORT file using the KEY = option with an

- unspecified or illegal (too long) Key value.
- RELEASE "T0".

11

MISSING OR DUPLICATE KEY

The program or task has attempted to:

- READ from a DIRECT or SORT file using the KEY = option when the specified Key value does not exist for any record in the file.
- WRITE to a DIRECT or SORT file using the KEY = option when the specified Key value already exists for a record in the file.

(Note that the DOM = option for an I/O directive provides a selective error branch for the ERR = 11 condition.)

12

UNDEFINED OR DUPLICATE FILE ID

The program or task has attempted to:

- OPEN, ADD, DROP or ERASE a disk file or program file with a file-id which is not found on an available disk volume—either a) not defined, or b) located on a DISABLED disk volume.
- OPEN a terminal device which has not been configured for the system.
- Define a disk file with a file-id which already exists on an available disk volume.
- Define a disk file or program using one of the reserved two-character device or task names, (i.e., "LP", "P1" ... "P9", "D1" ... "D9", "SY", "T0" ... "TF", "GO" ... "GN").

13

FILE/DEVICE ACCESS

The program or task has attempted to:

- Execute an input directive (INPUT, READ, EXTRACT, or FIND) from an output-only terminal device such as a printer.
- Execute an output directive (WRITE or PRINT) to an input-only terminal device.
- WRITE or PRINT to a DIRECT or SORT file where the record to be affected is not specified by IND = or KEY = or by the record having been EXTRACTed by this task.

- Execute an INPUT directive from a disk file using syntax elements which are allowed only in INPUT from a terminal:
 - Data Positioning
 - Mnemonics
 - Output String
 - Verification Modifier
- Communicate between the User Task and a Ghost Task if both tasks are in the same mode (input or output mode) concurrently.
- ADDR a non-program file to the Public Directory.
- DROP a program that is busy.
- DROP a terminal device.

14

FILE/DEVICE USAGE

The program or task has attempted to:

- OPEN a terminal device which is currently OPENed for another task.
- OPEN any file or terminal device on a channel number which is currently OPENed by the same task.
- Execute an I/O directive with a channel number not currently OPENed by that task.
- Define a disk file on a logical disk volume previously DISABLEd by the same task (for some operating systems).
- DISABLE or ENABLE a logical disk volume already DISABLEd or ENABLEd (respectively) for the current task.
- LOCK a file which is not OPENed to the current task.
- LOCK or UNLOCK a file already LOCKed or UNLOCKed (respectively) by the current task.
- ADDR a program already in the Public Programming Directory.
- Execute a START directive with a task-id not configured for the system or with a device-id.

- 15 **DISK ALLOCATION**
The program or task has attempted to define a disk file (either a data file or a program file) in disk sectors already allocated to other disk files or to the system (for some operating systems).
- 16 **DISK DIRECTORY CAPACITY**
The program or task has attempted to:
- Define a disk file when the disk directory is full.
 - OPEN a file when the task's File Control Table is full.
- 17 **INVALID PARAMETER**
The program or task has attempted to:
- Reference a terminal device, task or logical disk number which is not configured for the system.
 - Execute a KEY = access to a file other than a DIRECT or SORT file.
 - Execute IOR, XOR or NOT directives with strings of unequal lengths.
 - SAVE, LOAD or RUN a non-program file.
 - LIST to or MERGE from a file other than an INDEXED file.
 - Execute an I/O directive to a program file.
- 18 **ILLEGAL CONTROL OPERATION**
The program or task has attempted to:
- START more tasks than the system is configured for.
 - RELEASE the System Control Task from itself (for some operating systems).
- 19 **PROGRAM FORMAT OR SIZE**
The program or task has attempted to:
- LOAD a program from a PROGRAM file which is empty.
 - LOAD a program whose internal name (in the disk file) does not match the PROGRAM file name.
-

- LOAD or RUN a program whose size exceeds the available User Task memory area.

STRUCTURE

20

STATEMENT STRUCTURE (SYNTAX)

The program or task has attempted to:

- Enter or execute a program statement with incorrect or missing punctuation, symbols or operators.
- Enter or execute a program statement with non-existent or incorrectly spelled directives or other syntax words.
- Enter or execute a program statement with incorrect or illegal variable syntax.
- Enter or execute a directive with illegal or conflicting options specified.
- Execute an EDIT directive with illegal parameters or syntax.
- Reference a hexadecimal data element with incorrect or illegal syntax; i.e., the syntax must:
 - Contain an even number of characters
 - Contain only numeric characters 0-9 and alphabetic characters A-F.
 - Be enclosed in dollar signs (\$... \$) unless used in a TABLE statement.
- Execute a directive reserved for the Control Task.

21

STATEMENT NUMBER ERROR

The program or task has attempted to:

- Reference a statement number which is not a positive integer between 1 and 65534.
- Execute an EDIT or DELETE directive on a non-existent statement number.
- Enter a PROGRAM MODE only statement without a statement number.

24

FUNCTION NAME DEFINITION

The program or task has attempted to define a Programmable Function DEF FNx or DEF FNx\$ with the same identification character (x) as

another Programmable Function existing in the same task.

25

UNDEFINED FUNCTION

The program or task has attempted to reference a Programmable Function DEF FNx or DEF FNx\$ which has not been defined within the current task.

26

VARIABLE USAGE

The program or task has attempted to:

- Specify a non-numeric character in a numeric data element. (Note that the converse—specifying a numeric in a string data element—is not an error since a numeric character is a string character also by definition.)
- Enter or execute a directive or function with a variable name of the wrong type (numeric or string) for that implied or required by that directive or function.
- Specify a non-integer for an integer-only numeric data element or parameter.

27

RETURN WITHOUT GOSUB

The program or task has attempted to:

- Execute a RETURN directive without an active GOSUB or SETESC directive.
- Execute an EXITTO directive without an active GOSUB or FOR directive.
- Execute a RETRY directive without an active SETERR or ERR = directive.
- Execute an EXIT directive from a program which is not a Public Program.
- EDIT or DELETE a statement which contains an active GOSUB, FOR, SETERR, or ERR = directive.

28

NEXT WITHOUT FOR

The program or task has attempted to execute a NEXT directive without an active FOR directive.

29

UNDEFINED MNEMONIC CONSTANT

The program or task has attempted to:

- Execute an I/O directive with a Mnemonic Constant which is undefined or illegal for the referenced terminal device.
- Execute an I/O directive with a Data Positioning Modifier which is undefined or illegal for the referenced terminal device.
- Specify an ESCape code (\$1B\$) in a data element.

LIMIT

30

PROGRAM CHECK SUM ERROR

The program or task has attempted to LOAD, RUN, LIST or execute the LST function on an invalid string.

31

INTERNAL STACK OVERFLOW

The program or task has attempted to:

- Execute a program with too many “nested” GOSUBs or FOR/NEXT loops.
- Compile a statement with too many “nested” parentheses.

32

RECORD TOO LARGE FOR BUFFER

The program or task has attempted to execute an I/O directive with a record size larger than the available buffer memory area.

33

MEMORY CAPACITY

The program or task has attempted to:

- Enter, EDIT or MERGE a statement(s) into the current program whose length causes an overflow of the User Task memory area.
- CALL a program whose length exceeds the available User Task memory area.
- START a task whose size exceeds the available memory in the memory bank.

35

PARENTHETIC EXPRESSION LIMIT

The program or task has attempted to execute a statement of excessive complexity. (The

primary cause is too many levels of nested parentheses.)

36

CALL/ENTER MISMATCH

The program or task as attempted to:

- Execute an ENTER directive in a Public Program with a variable list which does not match (either in number or type of variables) the variable list of the CALL directive which CALLED the program.
- Execute an ENTER directive more than once in a Public Program.
- Execute an ENTER directive in a non-Public Program.

38

ILLEGAL COMMAND IN A PUBLIC PROGRAM

The program or task has attempted to:

- Execute an EXECUTE, LIST, RUN, DELETE, MERGE or SAVE directive in a Public Program.
- Re-DIMension a numeric array in a Public Program which has passed through the ENTER directive.

39

ESCAPE IN A PUBLIC PROGRAM

The program or task has attempted to:

- Execute an ESCAPE directive in a Public Program.
- Execute an ESCape key on the Task VDT during execution of a Public Program.

EXECUTION

40

NUMERIC VALUE OVERFLOW

The program or task has attempted to:

- Execute an arithmetic operation resulting in a numeric element of $10 \wedge 128$ or greater.
- Assign the result of a BIN function to a string data element which is not long enough.

INTEGER RANGE

The program or task has attempted to:

- Reference an I/O channel with an integer outside the range 0 to 9.
- Reference a logical disk number with an integer outside the range 0 to 9.
- Reference a file size (number of records) or an INDEX (record number) with an integer outside the range 1 to $(2 \wedge 23) - 1$.
- Reference a file record size (number of bytes) with an integer outside the range 2 to 32767.
- Reference a Data Positioning parameter with an integer outside the range 0 to 255.
- Reference a string subscript in a DIM Directive or a Substring with an integer outside the range 1 to 32767.
- Reference a subscript of a numeric array in a DIM Directive or variable reference with an integer outside the range 0 to 4095.
- Define a program size (number of bytes) with an integer outside the range 2 to 32767.
- Execute a PRECISION directive with an integer outside the range 0 to 14.
- Execute an ON n GOTO directive where n is an integer greater than 65535.
- Reference an exponentiation operation (raising to a power) where the exponent is an integer greater than 32767.
- Define the KEY field length of a DIRECT or SORT file with an integer outside the range 1 to 54 (or 1 to 56 if there are less than 32768 records in the file).
- Reference a step-value in a POS function with an integer outside the range 1 to 32767.
- Reference a CHR function with an integer argument outside the range 0 to 255.

42

NONEXISTENT SUBSCRIPT

The program or task has attempted to:

- Reference an element of a numeric array which has not been DIMensioned.
- Reference an element of a numeric array with a subscript(s) outside the DIMension limits.

44

STEP SIZE ZERO

The program has attempted to execute a FOR/ NEXT Directive with a STEP size of 0.

45

STATEMENT USAGE

The program or task has attempted to:

- Enter a CONSOLE MODE only statement with a statement number (indicating PROGRAM MODE usage).
- Execute a PROGRAM MODE only statement in the CONSOLE MODE.
- Enter or execute a LIST or DELETE directive with a starting statement number specified greater than the ending statement number.
- Execute an I/O Directive with an IOL= option referencing a statement which is not an IOLIST.
- Execute an I/O Directive with a TBL= option referencing a statement which is not a TABLE.

46

STRING SIZE

The program or task has attempted to:

- Execute an I/O Directive with a KEY= option where the specified KEY is longer than the defined KEY length for the file.
- Execute a SETDAY Directive with a string argument greater or less than 8 characters long.
- Execute an ASC Function with the null string ("") specified as the argument.

47

INVALID SUBSTRING REFERENCE

The program or task has attempted to reference a substring character position beyond the length of the specified string.

48

INPUT VERIFICATION

The program or task has attempted to:

- INPUT a numeric data element outside the VERIFICATION limits specified in the INPUT Directive.
- INPUT a string data element which does not match a string constant specified for VERIFICATION branching (if specified) and whose length does not fall within the LEN = limits specified in the INPUT directive.

103

UNEXPECTED OPERATING SYSTEM ERROR

The program or task has attempted to perform some operation, or some condition has occurred which is not covered by the above Error Processing Codes and which may be unique to a particular operating system.



CHAPTER NINE—THOROUGHbred/OS UTILITIES

Thoroughbred/OS includes a number of powerful Utility Programs which permit the user to perform commonly required maintenance functions on programs and files.

Following are the names and functions of the BASIC Utility Programs:

Program Name	Short Name	Function
**PSD	**	Utility Selector Menu and Maintenance Program
*1PSD	*1	Initialize Device
*2PSD	*2	Copy BASIC
*3PSD	*3	Copy Diskette
*4PSD	*4	Add/Change Directory Names
*APSD	*A	Print Disk Directory
*BPSD	*B	Build FileSet List
*CPSD	*C	Compare Programs
*DPSD	*D	Rename Programs/Files
*EPSD	*E	Erase Programs/Files
*FPSD	*F	Display File Parameters
*GPSD	*G	Ghost Task Communication
*HPSD	*H	Display Programs
*IPSD	*I	Hexadecimal File Dump
*JPSD	*J	Select Disk Environment
*KPSD	*K	Print Program Listing
*LPSD	*L	Display Disk Directory
*MPSD	*M	Display Task Allocation
*NPSD	*N	BASIC Configurator
*OPSD	*O	Define Files
*PPSD	*P	Renumber Programs
*QPSD	*Q	String Search and Replace (for programs)
*SPSD	*S	Transfer Programs/Files
*TPSD	*T	File Transfer and Expand
*VPSD	*V	Diskette Backup/Restore
*XPSD	*X	Channel Cross Reference
*YPSD	*Y	List Configuration
BASIC		Exit to BASIC Console Mode

From BASIC, enter RUN "***PSD" to produce the BASIC Utilities Menu. From the menu program, enter either the program name or the program number to select a particular Utility Program. (Note: Some systems allow these Utility Programs to be abbreviated to the first two characters, e.g., RUN "***PSD" is the same as RUN "***".)

The BASIC Utilities use the Control or Function keys of the user terminal with the following standardized function assignments:

Control Key or Function Key		Function
CTL-I	or F1	"Y" to YES/NO questions; otherwise, same as 'CR'.
CTL-II	or F2	"N" to YES/NO questions.
CTL-III	or F3	Return to previous question.
CTL-IV	or F4	Restart Utility or return to Selector.

NOTE: Certain Thoroughbred/OS Utilities are common only to specific computers. Please consult your manufacturer's manual to determine the utilities that are applicable to your system.

****PSD**

UTILITY SELECTOR MENU AND MAINTENANCE PROGRAM

This program displays the Utility menu on the terminal and asks the user to select the Utility program to be run. It may also be used to add and delete Utilities to and from the Utilities Menu. A maximum of 34 Utilities is allowed.

To invoke this Utility:

ENTER: RUN "*PSD"**

The BASIC Utilities Menu is then displayed with the following prompt:

ENTER SELECTION OR CR TO END

Enter the name or number of the Utility to be run or depress 'CR' to end the Utility selector. See the following sections for adding or deleting a Utility.

Adding a Utility

Enter @ADD in response to the ENTER SELECTION prompt to add a Utility program to the menu. The system will respond with the following prompt:

NAME:

Enter the name of the Utility program to be added. The system will then respond with the following:

DESCRIPTION:

Enter a description of the Utility program to be added. If the length of the Utility name, plus the description, is more than 32 characters, the description is truncated.

The system issues the following notification:

INITIALIZING CONTROL FILE

and, redisplay the BASIC Utilities Menu with the new Utility listed.

Deleting a Utility

Enter @DELETE in response to the ENTER SELECTION prompt to remove a Utility from the menu. The system will respond with the following prompt:

NAME

Enter the name of the Utility to be removed from the list. The system will redisplay the BASIC Utility Menu with the Utility removed.

ERROR MESSAGES

INVALID SELECTION

Either a selection greater than the number of Utilities in the menu, or the name of a non-existent Utility program was entered. Enter another valid selection.

nnnnnn IS ALREADY IN THE UTILITY MENU

The named Utility program already exists and, therefore, cannot be added.

nnnnnn IS NOT IN THE UTILITY MENU

The named Utility program does not exist and, therefore, cannot be deleted.

CAN'T OPEN CONTROL FILE '*UTIL'!! ERR = xx

The Utility program could not access the selector control file which contains the names and description of all the Utility programs. The run is aborted.

CONTROL FILE IS FULL

The control file already contains the maximum 34 Utilities and another Utility program may not be added.

ERROR xx HAS OCCURRED, CR TO CONT OR CTL4 TO END

An unexpected error has occurred, probably I/O related.

1PSD*INITIALIZE DEVICE**

This Utility program is used to initialize hard or floppy disks. Up to 30 directories or logical disks may be created for the disk type you are initializing. In addition, this Utility provides a floppy disk formatting option which is used for initializing a brand new floppy disk. To invoke this Utility:

ENTER: *1PSD and 'CR'

This entry may also be entered as *1 or as the function number indicated on the BASIC Utilities Menu. The system will call up the *1PSD—INITIALIZE DEVICE screen with the following message displayed:

**THIS PROGRAM WILL DESTROY ALL INFORMATION
ON THE SELECTED DEVICE!
IS THIS WHAT YOU WANT? (Y/N)**

ENTER: appropriate response and 'CR'

Disk initialization destroys any information previously residing on the disk. Therefore, use caution when running this Utility. Make sure that the disk you have selected to initialize does not contain any information that should not be destroyed. Enter N to end the procedure and return to the BASIC Utilities Menu. Enter Y to continue; the system responds with the following prompt:

ENTER DEVICE NAME (H0, F0, F1) ('CR' = F0)

ENTER: appropriate device and 'CR'

Enter the device type that you are initializing: H0 for hard disks; F0 or F1 for floppy disks. F0 is the default. The system then responds with the following message:

ENTER DEVICE LABEL ('CR' FOR *MASTER*)

ENTER: xxxxxxxx and 'CR'
(where: xxxxxxxx is a one to eight character label)

This entry assigns a label to your disk. The default disk label is *MASTER*. When a label name has been entered, the system will respond with the following:

**ENTER NUMBER OF SECTORS FOR AVAILABLE
SECTOR TABLE ('CR' = 10)**

ENTER: appropriate value and 'CR'

This entry allocates space for the Available Sector Table (a list of sectors available for use). The default value is 10, which is sufficient in most cases. If, however, you are initializing a rather large disk, you may wish to allocate additional space for the table; up to 30 sectors may be requested. When a value has been entered, the system responds with the following message:

SHOULD BASIC BE INCLUDED? (Y/N)

ENTER: appropriate response and 'CR'

Enter N to not include BASIC on the new disk. Enter Y to include BASIC, and the system will respond with the following:

FROM WHICH DEVICE ('CR' = H0)

ENTER: appropriate response and 'CR'

Enter the device *from* which you are copying BASIC. H0 is the default. When you have entered the device, the system responds with the following prompt:

ENTER DIRECTORY NAME ('CR' = FD30)

ENTER: xxxx and 'CR'

(where: xxxx is a one to four character name)

Up to 30 directory names can be entered, ten of which can be accessed at any one time. Special characters may be used, but not spaces. The default directory name is FD30. For each directory name entered, the system responds with the following:

NUMBER OF FILES ('CR' = 250)

ENTER: appropriate number and 'CR'

Enter the number of files to reside on the directory. The default value is 250. After you have entered the number of files for a directory, the system returns you to the 'ENTER DIRECTORY NAME' prompt. After having created at least one directory, the prompt appears as follows:

ENTER DIRECTORY NAME ('CR' = END)

Depress 'CR' when you have created all the directories that you want and wish to continue with the initialization procedure. The system will display a summary of the directories along with the following question:

IS THE ABOVE INFORMATION CORRECT? (Y/N)

ENTER: appropriate response

Enter N if any of the information is not correct. The system will return you to the first question of this Utility to redo the procedure. Enter Y if the information is correct and you wish to continue with the procedure. The system will respond with the following command:

LOAD DISKETTE TO BE INITIALIZED IN DRIVE Fn, 'CR' WHEN READY OR 'F4' TO END

Load the diskette into the appropriate drive and depress CR. The system responds with the following question:

DOES DISKETTE NEED TO BE FORMATTED? (Y/N)

ENTER: appropriate response

Enter Y only if this is a brand new floppy disk that has not been previously formatted. The system will respond with a 'FORMATTING' notification. When formatting is complete, the system automatically verifies it. When verification is complete or if you entered N in response to the question, the system responds with the following message:

INITIALIZING DIRECTORIES

And, if you selected BASIC to be included, the following message will also be displayed during initialization:

COPYING BASIC

When the procedure is complete, the system responds with the following message:

INITIALIZATION COMPLETE ('CR' TO END)

ENTER: 'CR'

The BASIC Utilities Menu will be redisplayed.

ERROR MESSAGES

ERROR FORMATTING DISK

Retry the entire procedure. If the error reoccurs, the floppy disk is probably damaged.

ERROR 103 HAS OCCURRED, 'CR' TO RETRY

Either the diskette was not inserted when instructed, or the drive door was not completely shut. Re-check these areas and correct error.

***2PSD**

COPY SMC BASIC

This Utility program enables you to copy SMC BASIC to either a hard disk or a floppy disk.

To invoke this Utility:

ENTER: *2PSD and 'CR'

This entry may also be entered as *2 or as the function number indicated on the SMC BASIC Utilities Menu. The system will call up the *2PSD—COPY SMC BASIC screen with the following prompt displayed:

SELECT 'DESTINATION' DEVICE 'H0' or 'F0'

ENTER: appropriate device

Enter the device *to* which you wish to copy SMC BASIC. H0 is a hard disk; F0 is a floppy disk. The system will respond with the following message:

COPY FROM 'x0' TO 'x0', IS THIS WHAT YOU WANT? (YES OR NO)

ENTER: appropriate response

If you selected 'H0' as your destination device, the question will read "FROM 'F0' TO 'H0' ". If you selected 'F0' as your destination device, the question will read, "FROM 'H0' TO 'F0' ". Enter N if anything is incorrect. The system will redisplay the beginning prompt. Enter Y to continue with the procedure. The system responds with the following question:

COPY CONFIGURATION FROM SOURCE DEVICE? (YES OR NO)

ENTER: appropriate response

Enter Y to copy the configuration to the destination device. Enter N if you do not want the configuration copied. The system will respond with the following command:

**LOAD [DESTINATION]/[SOURCE] DISKETTE IN DEVICE 'F0'
PRESS RETURN KEY WHEN READY**

'DESTINATION' will appear in this prompt if you selected 'F0' as your destination device. 'SOURCE' will appear in this prompt if you selected 'H0' as your destination device.

Load the diskette and depress 'CR'. The system will respond with a 'COPYING' notification. When the procedure is complete, the following message appears:

COPY COMPLETE (PRESS RETURN)

ENTER: 'CR'

The BASIC Utilities Menu is redisplayed.

ERROR MESSAGES

ERROR '103' HAS OCCURRED, 'CR' TO RETRY

Either the diskette was not inserted at the appropriate time, or the floppy door is not completely shut. Please re-check these areas and correct error.

This Utility program enables you to: (a) copy the contents of a floppy disk onto an area of the hard disk and then copy the contents of that area onto another floppy disk, or (b) copy from one floppy drive to a second floppy drive (if so configured).

To invoke this Utility:

ENTER: *3PSD and 'CR'

This entry may also be entered as *3 or as the function number indicated on the SMC BASIC Utilities Menu. The system will call up the *3PSD—COPY DISKETTE screen with the following command displayed:

**LOAD 'SOURCE' DISKETTE IN DEVICE 'Fn',
PRESS RETURN WHEN READY**

Load the diskette to be copied into the appropriate drive and depress CR. The system responds with a 'COPYING' notification. When the contents of the diskette have been copied, the system responds with the following command:

**LOAD 'DESTINATION' DEVICE IN DEVICE 'Fn',
PRESS RETURN WHEN READY**

Load the diskette to be copied and depress 'CR'. The system responds with the following question:

DOES DISKETTE NEED TO BE FORMATTED (Y/N)?

ENTER: appropriate answer

Only enter Y if the diskette is a brand new diskette. When this prompt is answered, the system automatically responds with a 'COPYING' notification. When the copy procedure is complete, the system responds with the following question:

COPY COMPLETE. DO YOU WANT ANOTHER?

ENTER: appropriate answer

Enter N if you do not want to make another copy. The system will then redisplay the BASIC Utilities Menu. Enter Y to make another copy; the system redisplay the following command:

**LOAD 'DESTINATION' DISKETTE IN DEVICE 'Fn',
PRESS RETURN WHEN READY**

Load the next diskette to be copied and repeat the procedure.

ERROR MESSAGES

CAN'T LOCATE SPACE ON THE HARD DISK FOR COPY AREA

This message indicates there is not enough space available on the hard disk to copy the contents of the floppy disk. Erase any unnecessary files and retry the procedure.

NEED TWO DEVICES TO BE ABLE TO COPY

Two disks are required in order to make a copy of the contents of a disk.

This Utility program allows you to add a new directory to a floppy or hard disk or to change the name of an existing directory on the floppy or hard disk. This Utility cannot delete directories or move directories from one disk to another.

To invoke this Utility:

ENTER: *4PSD and 'CR'

This selection may also be entered as *4 or as the function number indicated on the BASIC Utilities Menu. The system will call up the *4PSD—ADD/CHANGE DIRECTORY NAMES screen with the following question displayed:

ADD OR CHANGE DIRECTORY NAME (A/C)

ENTER: appropriate response and 'CR'

Adding a Directory

Enter A in response to the ADD OR CHANGE DIRECTORY NAME prompt to add a new directory. The system will respond with the following prompt:

ENTER DISK—HARD OR FLOPPY (H/F)

ENTER: appropriate response and 'CR'

Enter H if you are adding the directory to the hard disk. Enter F if you are adding the directory to the floppy disk. Note: If your system is dual drive, the following prompt will appear:

WHICH DRIVE (0/1)

ENTER: appropriate response and 'CR'

The system will display the existing directories, along with the following prompt:

ENTER NUMBER OF FILES

ENTER: desired number of files and 'CR'
(The minimum acceptable value is 16)

When a valid number is entered, the system responds with the following prompt:

ENTER DIRECTORY NAME

ENTER: a name from 1 to 4 characters and 'CR'

When a valid directory name is entered, the system will respond with the following message at the bottom of the screen:

PROCESS COMPLETE (CR TO CONTINUE)

Depress CR and the system redisplay the SMC BASIC Utilities Menu. If you want to immediately access the new directory, run the *JPSPD—SELECT DISK ENVIRONMENT Utility to change the accessible directories.

Changing a Directory

Enter C in response to the ADD OR CHANGE DIRECTORY NAME prompt to change a directory. The system responds with the following prompt:

ENTER DISK—HARD OR FLOPPY (H/F)

ENTER: appropriate answer and 'CR'

Enter H to change the name of a directory that exists on the hard disk. Enter F to change the name of a directory that exists on the floppy disk. The system displays a list of the directories, along with the following prompt:

ENTER OLD DIRECTORY NAME

ENTER: the directory name to change and 'CR'

The system will respond with the following prompt:

ENTER NEW DIRECTORY NAME

ENTER: the new directory name
(from 1 to 4 characters)

The system will respond with the following prompt:

xxxx (old directory name) = = = yyyy (new directory name)
OKAY (Y/N)

ENTER: appropriate answer and 'CR'

Enter N if the name change is incorrect. The system will respond with the following message:

PROCESS ABORTED (CR TO CONTINUE)

ENTER: 'CR'

The system will automatically redisplay the ADD OR CHANGE DIRECTORY NAME prompt. Retry the procedure or depress CTL-IV (F4) to return to the SMC BASIC Utilities Menu.

Enter Y if the name change is correct. The system responds with the following message:

PROCESS COMPLETE (CR TO CONTINUE)

ENTER: 'CR'

If the directory renamed was one of the ten accessible directories, the Utility will automatically disable the old directory and display the *JPSP—SELECT DISK DIRECTORIES screen. To access the newly-named directory, follow the applicable procedure described in the *JPSP Utility documentation.

If the directory renamed was not one of the ten accessible directories, the Utility will automatically redisplay the SMC BASIC Utilities Menu. If you want to immediately access the newly-named directory, run the *JPSP Utility to change the accessible directories.

ERROR MESSAGES

ALREADY DEFINED

The name you have selected to call a new directory, or to rename an existing directory, already exists.

HARD DISK NOT CONFIGURED

You have specified H0 for disk type and your system is a dual floppy system. Specify F0 or F1 as the disk type.

INVALID NAME

You are trying to enter an invalid name. Either it has a space as its first character or it is longer than 4 characters. Enter a valid name.

SAME NAME—NO CHANGE

You are entering the same name for the OLD DIRECTORY NAME and NEW DIRECTORY NAME prompts. Enter a different name.

TABLE OF CONTENTS IS ALREADY FULL

No more space is available to add a new directory. Either rename an existing directory or delete unnecessary directories by (a) first doing a selective backup using the *VPSD—DISKETTE BACKUP/RESTORE Utility, (b) then initializing the disk via *1PSD—INITIALIZE DEVICE (defining directory names only for those directories backed up), and (c) finally, restoring the backed-up directories via the *VPSD Utility.

***APSD**

PRINT DISK DIRECTORY

This Utility program prints the contents of a disk directory on a printer. For each file on the disk, the file name, file type, starting, ending, and total sectors, key size, record size, records in use, and available records are shown. In addition, for program files whose first statement is a REMark, that statement will be printed as the program ID. The listing may be sorted in either sector or alphabetical order.

To invoke this Utility:

ENTÉR: *APSD and 'CR'

This entry may also be entered as *A or as the function number indicated on the BASIC Utilities Menu. The system will call up the *APSD—PRINT DISK DIRECTORY screen with the following prompt displayed:

SELECT PRINTER: pp pp:

(where: pp is a configured printer.

Printers in use are shown in background mode.)

Enter the name of the printer where output is to be sent. Depress CR to send the output to 'LP'. If the requested printer cannot be opened, the user will be asked for another request. If only one printer is configured, this question will not be asked.

The system will then respond with the following prompt:

DISC ID

Enter a valid disk number or volume name. The system will respond with the following prompt:

BY AVAILABLE LOCATION, SECTORS OR NAME (A/L/N):

Enter A to print the available sectors for the disk. Enter L to sort the directory in sector order. Enter N to sort the directory in alphabetical order. If alphabetical order is selected, the message 'SORTING' will appear while the directory is sorted.

EXPLANATION OF FIELDS

NAME	Eight character file name.
TYPE—IND	Indexed files.
SER	Serial files.
SRT	Sort files.
DIR	Direct files.
PGM	Program files.
START SECTOR	Location of the beginning of the file on the disk.
END SECTOR	Last sector used by the file.
TOTAL	Number of sectors used by the file.
KEY	For sort and direct files, the length of the key in bytes; for other files, this field is 0.
RECORDS	Number of logical records in the file. For program files, this field is 1. For sort files, it is the number of keys.
BYTES/RECORDS	Record length in bytes; this field is 0 for sort files.
IN USE	Number of records used. For programs, this is the actual length of the program in bytes.
AVAILABLE	Number of records that are not being used. If an asterisk (*) appears to the right of this number, there are deleted keys that have been counted as used rather than as available.
PROGRAM ID	The contents of the first REM statement in a program, or "NONE" if the first statement is not a REM. If the file is a directory, 'DIRECTORY' will appear in this field.

NOTES

If the directory is printed in sector order, an entry consisting of **"*AVAILABLE*"** and starting, ending, and total sectors will be printed for each unused block on the disk.

ERROR MESSAGES

PRINTER IN USE

The requested printer is being used by another task; select another.

PRINTER NOT READY

The requested printer is offline; select another, or ready printer.

INVALID FILE NAME

The requested volume does not exist; enter another.

DISK nn NOT CONFIGURED

The disk number given is invalid or the disk does not have a Thoroughbred/OS directory, or it is disabled.

'xxxxxx' NOT A DIRECTORY

The volume named was opened and was not a directory. Enter another volume name.

CAN'T DEFINE WORK FILE ON DISK nn, ERROR = xx

The Utility was unable to create a file to sort the directory in alphabetical order. Control returns to the **'LOCATION, SECTORS OR NAME'** query.

ERROR xx HAS OCCURRED, CR TO CONT, CTL4 TO END

An unexpected error has occurred, probably I/O related.

This utility inputs a list of file names and masks (described below) from the terminal, compares the list with all files on a specified disk, and places all matching file names into a Sort file on the same disk. The Sort file is named CSYSFLT#, where CSYSFL is a constant and T# is the terminal ID.

To invoke this Utility:

ENTER: *BPSD and 'CR'

This entry may also be entered as *B or as the function number indicated on the BASIC Utilities Menu. The system will call up the *BPSD—BUILD *SRT screen with the following prompt displayed:

DISC ID:

Enter a valid disk number. If the directory for the specified disk cannot be opened, the user will be asked to supply another disk number. When a valid disk number is entered, the system will respond with the following prompt:

ENTER FILE NAME OR MASK: (@) FOR ALL, (CR) TO END

Enter up to 48 file names or masks. The 'at' sign (@) will copy all names in the directory into *SRT. If a file does not exist on the specified disk, or if it is not of the correct type, the name will be rejected. Depress CR to stop entering names and masks. Depress CTL-IV (F4) to end the Utility program without building *SRT. See Notes on masks below.

NOTES

Masks are one to eight character strings used as patterns to match file names. They contain alphanumeric characters plus the question mark (?) and 'at' sign (@) characters. A question mark will match any single character in a file name including trailing spaces. An 'at' sign may only appear as the last character in the mask. It will match the remainder of the file name. For instance, the mask '**?PSD@' will match all the Utility programs and their overlays. If the directory contains the following entries:

FM012 FM117 FM193 UD790 UD984 PM018 PM094 PM112 PM193
ZP ZP1 ZP117 ZP193

... the matches shown here would result:

FM@—FM012 FM117 FM193

?M0@—FM012 PM018 PM094

ZP?—ZP ZP1

??1@—FM117 FM193 PM112 PM193 ZP1 ZP117 ZP193

??193—FM193 PM193 ZP193

ERROR MESSAGES

LIMIT IS 48 NAMES/MASKS

The Utility will not accept any more file names or masks and will begin its directory search.

CAN'T DEFINE *SRT x, ERROR = xx

The Utility could not create a new *SRT to place file names in. The run is aborted.

ERROR xx HAS OCCURRED, CR TO RETRY

An unexpected error, probably I/O related, has occurred.

This Utility program compares two programs and produces a listing on the printer of all changes that have to be made to one program to make it identical with the other program. You may compare two programs residing on the same logical disk, or you may compare programs residing on different disks.

To invoke this Utility:

ENTER: *CPSD and 'CR'

This entry may also be entered as *C or as the function number indicated on the BASIC Utilities Menu. The system will call up the *CPSD—COMPARE PROGRAMS screen with the following prompt displayed:

SELECT PRINTER: pp pp:

(where: 'pp' is a configured printer.

Printers in use are shown in background mode.)

Enter the name of the printer where output is to be sent. A carriage return sends the output to 'LP'. If the requested printer cannot be opened, the user will be asked to request another. If only one printer is configured, this question will not be asked. When the printer name has been entered, the system will respond with the following prompt:

ENTER 'OLD' DIRECTORY OR FILE NAME:

Enter the directory name if you are comparing programs from different directories. Enter the name of the program if you are comparing programs residing on the same directory.

If you enter a directory name, the system will respond with the following prompt:

ENTER 'NEW' DIRECTORY NAME

Enter the name of directory containing the program that is to be compared with a program from the previously specified directory.

When you enter a valid name, the system will respond with the following prompt:

ENTER 'OLD' PROGRAM NAME

Enter the name of a program to be compared. If you are comparing programs residing on the same directory and have entered a program name in response to the first prompt, this field will have an automatic insertion.

You will be prompted again if the program name is not valid. When a valid program name has been entered, the system will respond with the following:

ENTER 'NEW' PROGRAM NAME

Enter the name of a program to be compared with the program specified for OLD PROGRAM. If the program name is not valid, the user will be prompted again.

NOTES

The Utility program detects three types of changes: (a) if a statement exists in the old program, but not in the new program, a delete line is printed, (b) if a statement exists in the new program, but not in the old program, an add line is printed, and (c) if a statement exists in both programs, but its contents have changed, a change message is printed. If no differences are found, the message 'PROGRAMS ARE IDENTICAL' will be displayed.

ERROR MESSAGES

PRINTER IN USE

The requested printer is being used by another task; select another.

PRINTER NOT READY

The requested printer is offline; select another.

nnnnnn IS UNINITIALIZED OR BLANK

The specified program is empty and cannot be compared. Enter another program name.

ERROR xx HAS OCCURRED, CR TO RETRY

An unexpected error has occurred, probably I/O related.

PURPOSE

This Utility program changes the names of programs or files without changing their file characteristics or location on the disk.

To invoke this Utility:

ENTER: *DPSD and 'CR'

This entry may also be entered as *D or as the function number indicated on the BASIC Utilities Menu. The system will call up the *DPSD—RENAME PROGRAMS/FILES screen with the following prompt displayed:

ENTER NAME

Enter the program or file name to be renamed. If the file cannot be opened, the user will be asked to supply another name. The system will then respond with the following prompt:

= =

Enter the new program or file name. The system will respond with the following message:

'xxxxxx HAS BEEN RENAMED yyyyyy'

(where: xxxxxx is the former name of the program
and yyyyyy is the new name.)

NOTES

This Utility renames a file by erasing the original file name and defining a file with the new name at the same location.

ERROR MESSAGES

nnnnnn ALREADY EXISTS ON DISK xx, CAN'T RENAME

A file with the new name already exists on the same disk as the file to be renamed. Enter another name.

nnnnnn ALREADY EXISTS ON DISK xx. CR TO CONTINUE

A file with the new name already exists on another disk. Enter CR to rename the file anyway, or CTL-III (F3) to enter another name.

**CAN'T CREATE NEW FILE, OLD ERASED, WAS AT SECTOR xx,
ERROR = YY**

The Utility was able to erase the old file, but could not create the new file. The old file sector is given so that the user can recover the file.

EPSD*ERASE PROGRAMS/FILES**

This Utility program erases groups of files and/or programs from a specified disk.

To invoke this Utility:

ENTER: *EPSD and 'CR'

This entry may also be entered as *E or as the function number indicated on the SMC BASIC Utilities Menu. The system will call up the *EPSD—ERASE PROGRAMS/FILES screen with the following prompt displayed:

ERASE PROGRAMS (P), FILES (F), OR BOTH (B)

Enter 'P' if only programs are to be erased. Enter 'F' if only data files are to be erased. Enter 'B' if both data files and program files are to be erased. The system will respond with the following prompt:

DISC ID:

Enter the number of the directory on which the specified files reside. When a valid directory number has been entered, the system will respond with the following:

ENTER FILENAME OR MASK; (@) FOR ALL, (CR) TO END

Enter up to 48 file names or masks. The 'at' sign (@) will copy all names in the directory into '*SRT'. If a file does not exist on the specified disk or if it is not of the correct type, the name will be rejected. Depress 'CR' to stop entering names and masks. Depress CTL-IV (F4) to end the Utility without building '*SRT'. To continue with the procedure, depress 'CR'. The system will respond with the following prompt:

VERIFY (Y/N):

Enter N and the file will automatically be erased. Enter Y and the Utility will ask, for each file, whether or not that file is to be erased, as follows:

ERASE nnnnnn (Y/N)

Enter Y to erase the named file or program. Enter N to retain this file and proceed to the next one. Depress CTL-IV (F4) to end the Utility. After each selection is made, the system responds with the following message:

xx FILES ERASED (HIT CR)

The Utility has erased all selected files. Depress CR to enter another selection or CTL-IV (F4) to end the Utility.

NOTES

This Utility calls the *BPSD Utility to obtain file names. The Utility will only erase files from one disk at a time. It disables other disks which contain files with the same name as the one that is erasing. At the end, all disks that the Utility program disabled will be enabled. The names of all the files erased are displayed on the terminal.

ERROR MESSAGES

nnnnnn DOES NOT EXIST

The named program or file could not be found and, thus, could not be erased. The Utility continues with the next file.

nnnnnn IS CURRENTLY OPEN, CAN'T ERASE

The named program or file is being accessed by another task and cannot be erased. The Utility continues with the next file.

LIMIT IS 48 NAMES/MASKS

The Utility will not accept more than 48 file names or masks and will begin its directory search.

ERROR xx ERASING nnnnnn

An unexpected error occurred while trying to erase the named file or program. The Utility continues with the next file.

CAN'T OPEN 'SRT'! ERR = xx**

The Utility was unable to obtain the file names gathered by *BPSD.
The run is aborted.

ERROR xx HAS OCCURRED, CR TO CONT OR F4 TO END

An unexpected error has occurred, probably I/O related.

This Utility program displays file characteristics for all types of files. For all files, name, type, disk, starting sector, ending sector, and total number of sectors are displayed. In addition, for indexed and serial files, record size and number of records are shown; for sort and direct files, key size, record size, number of records, records in use, and available records are shown; and for program files, defined size, actual size, available bytes, and program ID are shown. The program ID is obtained from the first line of the program if it is a REM statement.

To invoke this Utility:

ENTER: *FPSD and 'CR'

This entry may also be entered as *F or as the function number indicated on the BASIC Utilities Menu. The system will call up the *FPSD—DISPLAY FILE PARAMETERS screen with the following prompt displayed:

ENTER FILE NAME (CR TO END):

Enter the name of the file whose characteristics are to be displayed. If the file cannot be opened, the user will be asked to supply another file name. Depress 'CR' to end the Utility. When a valid file name is entered, the system will respond with the following prompt:

HARD COPY (Y/N)

Enter N if a hard copy is not desired. The ENTER FILE NAME prompt is returned. Depress F4 to return to *FPSD—DISPLAY FILE PARAMETERS screen, then enter CR to terminate the Utility. Enter Y to have the file characteristics that are displayed on the terminal sent to a printer. The system will respond with the following prompt:

SELECT PRINTER: pp pp:

(where: 'pp' is a configured printer.

Printers in use are shown in background mode.)

Enter the name of the printer where the output is to be sent. Depressing CR sends the output to 'LP'. If the requested printer cannot be opened, the user will be asked to select another. If only one printer is configured, this question will not be asked.

NOTES

When sort or direct files have deleted keys, the Utility will display the message 'CHASING DELETED RECORDS' while it counts records in order to display accurate statistics.

ERROR MESSAGES

ERROR 0 HAS OCCURRED, PRINTER PROBABLY OFFLINE

Check if printer is not ready or is out of paper. Depress 'CR' to continue.

ERROR xx HAS OCCURRED. 'CR' TO RETRY, CTL-II FOR NEXT, CTL-IV TO END

An unexpected error has occurred, probably I/O related. Depress CR to retry, CTL-II (F2) to enter a new file name, or CTL-IV (F4) to terminate the Utility.

PRINTER IS IN USE

The requested printer is being used by another task; select another.

PRINTER NOT READY

The requested printer is offline; select another.

UNABLE TO OPEN FILE: xxxxxxxx

The file named does not exist. Enter another file name.

This Utility program connects your terminal to Ghost Task Channel 0 and enables you to interface with a specified Ghost Task. This Utility can be used either to initiate a Ghost Task or to look at a Ghost Task that is currently executing. If the task running as a Ghost Task has stopped, due to an error condition or completion of program execution, the Utility allows you to access and modify it through console mode.

To invoke this Utility:

ENTER: *GPSD and 'CR'

This entry may also be entered as *G or as the function number indicated on the BASIC Utilities Menu. The system will call up the *GPSD—GHOST TASK COMMUNICATION screen with the following prompt displayed:

ENTER GHOST ID

ENTER: appropriate response

Enter the Ghost ID (G'n') with which you want to interact. If the task can be found, the following message will appear:

**COMMUNICATING WITH G'n', G'n' WILL NOW BE ATTACHED TO
T'n' PRESS THE 'ESCAPE' KEY TO BREAK**

(where: G'n' is the ghost task ID and T'n' is the terminal ID)

Depress the ESCAPE key to end Ghost Task Communication. The system will respond with the following message:

THE ATTACHMENT BETWEEN G'n' and T'n' HAS BEEN BROKEN
(where: G'n' is the Ghost Task ID and T'n' is the terminal ID)

The system will redisplay the BASIC Utilities Menu.

ERROR MESSAGES

NO GHOST ARE CONFIGURED

This message indicates that there are no Ghost Tasks defined for your system. Use the *NPSD BASIC CONFIGURATOR Utility, and select SYSTEM OPTIONS to modify your system.

TASK IS NOT CONFIGURED—‘CR’ TO CONTINUE

This message indicates that the Ghost Task specified is not configured for the system. To see the Ghost Task(s) configured for the system, use the *NPSD BASIC CONFIGURATOR Utility and select SYSTEM OPTIONS.

TASK IS NOT STARTED—‘CR’ TO CONTINUE

This message indicates that the Ghost Task specified has not been initiated. Initiate the desired Ghost Task using the START directive (refer to the Chapter on System Directives).

***HPSD**

DISPLAY PROGRAMS

This Utility program lists programs on the display terminal, one screen at a time. Listing may begin at a specified statement or at the occurrence of a specified search string.

To invoke this Utility:

ENTER: *HPSD and 'CR'

This entry may also be entered as *H or as the function number indicated on the BASIC Utilities Menu. The system will call up the *HPSD—DISPLAY PROGRAMS screen with the following prompt displayed:

PROGRAM:

Enter the name of the program to be listed. If the file cannot be opened or if it is not a program, the user will be asked to supply another program name. The system will respond with the following prompt:

**ENTER LINE NUMBER, CR TO CONTINUE,
CTL-1 TO SEARCH, OR CTL-4 TO END**

This message is displayed at the bottom of each screen. Depress CR to display the next screen. Enter a statement number to start the next screen at that statement, depress a CTL-1 (F1) to start the next screen at the occurrence of a search string, or enter HC to write the current screen to a specified printer.

If you depress CTL-1 to search, the system issues the following prompt:

SEARCH STRING [(CR = 'xxxx')]

Enter the string that the Utility is to search for. If there has been a previous search, the last search string will be shown. Depressing CR displays the next occurrence of the old search string. After entering the string, the system responds with the following prompt:

LINE NUMBER TO BEGIN SEARCH (CR = BEGINNING)

Enter the statement number at which the Utility is to begin searching. Depress CR to begin searching at the first statement in the program.

ERROR MESSAGES

nnnnnn IS UNINITIALIZED OR BLANK

The requested program is empty, enter another program name.

ERROR xx HAS OCCURRED. (CR) TO RETRY

An unexpected error has occurred, probably I/O related.

***IPSD**

HEXADECIMAL FILE DUMP

This Utility program produces hexadecimal dumps of files on either the terminal or a printer. Records are dumped in both hex and ASCII. This Utility also allows the user to look at all data in files and programs and at the key index area of sort and direct files.

To invoke this Utility:

ENTER: *IPSD and 'CR'

This entry may also be entered as *I or as the function number indicated on the BASIC Utilities Menu. The system will call up the *IPSD—HEXADECIMAL FILE DUMP screen with the following prompt displayed:

HARD COPY (Y/N):

Enter Y to send the dump to a printer. Enter N to display the dump on the terminal. The system will respond with the following prompt:

FILE NAME:

Enter the name of the file to be dumped, or depress CR to end the Utility. If the specified file cannot be opened, the user will be asked to supply another file name.

The system will generate different prompts according to whether the dump is to be displayed on the terminal or sent to a printer. The following list outlines the prompts generated:

OUTPUT TO TERMINAL

RECORD NUMBER

Enter the number of the record to be dumped. Depressing CR will dump the next record in sequence. Record numbers start with zero.

RECORD NUMBER/ 'KEY'/'KIA'

For direct files, enter KIA to dump the key index area. Enter KEY if records are to be chosen by their key values. Record numbers may be entered just as in the previous RECORD NUMBER prompt.

OFFSET (*256 BYTES) If the record size is greater than 256 bytes, it cannot be displayed in one piece on the terminal. Enter the offset of the 256 byte block to be dumped. Depressing CR will dump the next block in sequence. The first 256 byte block is block zero.

KEY Enter the key of the record that is to be dumped.

SECTOR Enter the index of the sector within the key index area that is to be dumped. Depressing CR will dump the next sector in sequence. The first sector is zero.

OUTPUT TO PRINTER

SELECT PRINTER Enter the name of the printer where output is to be sent. Depress CR to send it to 'LP'. If only one printer is configured, this question will not be asked.

RECORD (R),KIA (K) For direct files, enter R to dump the data area. Enter K to dump the key index area.

FROM RECORD Enter the number of the first record to be dumped to the printer. Depressing CR will set it to zero.

TO RECORD Enter the number of the last record to be dumped to the printer. Depressing CR will set it to the last record in the file.

FROM SECTOR Enter the index of the first sector of the KIA to be dumped to the printer. Depressing CR will set it to zero.

TO SECTOR Enter the index of the last sector to be dumped. Depressing CR will set it to the highest numbered sector in the KIA.

NOTES

Each record or sector is dumped in blocks of up to 256 bytes in a matrix of up to 16x16 hexadecimal characters. To the right of this matrix are the equivalent ASCII character strings. Non-printing characters are shown as periods.

ERROR MESSAGES

PRINTER IN USE

The requested printer is being used by another task. Select another.

PRINTER NOT READY

The requested printer is offline. Select another.

KEY NOT FOUND IN FILE: kkkkkk

The specified key is not in the file, enter another key.

ERROR xx HAS OCCURRED, CR TO RETRY

An unexpected error has occurred, probably I/O related.

This Utility program allows the user to select, from all the directories configured on the hard disk and any floppy disk(s) loaded, the directories to which access is desired. If a floppy containing desired directories is not loaded when invoking this Utility, an option is provided that allows you to load one. A maximum of ten directories may be accessed at any one time. This Utility also allows you to disable/enable selected directories.

To invoke this Utility:

ENTER: *JPSD and 'CR'

This entry can also be entered as *J or as the function number indicated on the Utilities menu. The system will call up the *JPSD—SELECT DISK ENVIRONMENT screen. The following is a sample screen:

***JPSD—SELECT DISK DIRECTORIES**

DISC DIRECTORY

0 H0 UTIL
1 H0 PROG
2 H0 DATA
3 H0 DOCU
4 H0 DEMO
5 H0 SURV
6 H0 VOL1
9 F0 MBOS

ENTER: 'D' TO CHANGE DISKETTE
 'S' TO CHANGE DIRECTORY
 OR DISC NUMBER TO CHANGE

' ' = ENABLED DISC
'D' = DISABLE DISC

'L' = LOCAL DISABLE
'R' = RESERVED DISK

NOTE: Directories 7 and 8 are not configured in the sample screen; thus, only 8 directories may be accessed at any one time. To configure directories, see *NPSD—BASIC CONFIGURATOR, Disk Directories.

EXPLANATION OF FIELDS

'D' TO CHANGE DISKETTE

This selection allows you to load a diskette containing directories you wish to access. This option can only be entered from the control terminal.

'S' TO CHANGE DIRECTORY

This selection allows you to select directories from those available on the hard disk, and, if a floppy is already loaded, from those on that floppy. This option can only be entered from a control terminal.

DISC NUMBER TO CHANGE

Allows you to locally disable/enable a directory.

' ' = ENABLED DISK

A directory followed by a blank space is enabled, i.e., all tasks can access the directory.

'D' = DISABLED DISK

A directory followed by 'D' is disabled, i.e., access to the directory is restricted for all users.

'L' = LOCAL DISABLE

A directory followed by 'L' is locally disabled, i.e., access to the directory is only restricted for the task issuing the LOCAL DISABLE. All other tasks have access to it.

'R' = RESERVED DISK

A directory followed by 'R' is reserved, i.e., the directory can only be accessed by the task issuing the RESERVE; no other tasks have access to it.

'S' TO CHANGE DIRECTORY

If the directories you want to access exist on either the hard disk or on a floppy already loaded, enter S to change the directories allowing access. NOTE: This entry can only be made from a control terminal.

The system will respond with a screen listing all the directories configured for the hard disk, and, if a floppy was previously inserted, for the floppy disk. The following is a sample screen:

*JPSD—SELECT DISK DIRECTORIES

DISC	DIRECTORY	EXISTING DIRECTORIES ARE:	
0	H0 UTIL	H0—UTIL	F0—FD30
1	H0 PROG	H0—PROG	F0—LIBR
2	H0 DATA	H0—DATA	
3	H0 DOCU	H0—DOCU	
4	H0 DEMO	H0—DEMO	
5	H0 SURV	H0—SURV	
6	H0 VOL1	H0—VOL1	
9	F0 MBOS	F0—MBOS	

ENTER DISC NUMBER TO CHANGE

ENTER: disc number

Enter the number of a directory that you wish replaced with a different directory. The system responds with the following prompt:

ENTER DIRECTORY NAME

Enter the name of the directory you want to access. If duplicate directory names exist for the hard disk and the floppy disk, the system will issue the following prompt:

WHICH DISK—H0 OR F0?

Enter the device where the directory resides. The system will replace the directory located at the number previously specified, with the new directory name entered. The system will then redisplay the 'ENTER DISC NUMBER TO CHANGE' prompt. Depress CR to return to the BASIC Utilities Menu or enter another directory to change.

'D' TO CHANGE DISKETTE

Enter D to access directories from a floppy not yet loaded in the drive. NOTE: This selection can only be made from a control terminal. The system responds with the following prompt:

ENTER UNIT 'CR' FOR F0

Enter the unit that you are loading. Depress CR for a default selection of a floppy. The system responds with the following command:

'CR' WHEN DISKETTE IS MOUNTED

Load the diskette and depress CR. The system responds with a screen listing the existing directories. To change a directory, follow the procedure outlined previously.

DISABLING/ENABLING A DISC

To disable or enable a disc, enter the number of the disc and depress CR. The appropriate letter is placed to the right of the directory name. If the directory was previously enabled, it becomes locally disabled. If the directory was previously locally disabled, it becomes enabled. Depress CR to return to the SMC BASIC Utilities Menu.

ERROR MESSAGES

A RESERVED OR DISABLED DISK CANNOT BE CHANGED

A Reserved or Disabled disk can only be changed using the ENABLE or RELEASE directive. (See the chapter on DISK AND FILE OPERATIONS.)

DISKETTE AND DIRECTORY CHANGES CAN BE MADE FROM CONTROL TERMINAL ONLY

A selection of 'D' or 'S' can only be made at the control terminal.

ERROR HAS OCCURRED AT STATEMENT 'xx' 'CR' TO RETRY

An unexpected error has occurred while trying to change a directory. Retry the procedure.

ONLY CONFIGURED DISKS MAY BE CHANGED

The disk you are trying to access is not configured. See the *NPSD Utility for configuring disks.

ONLY EXISTING DIRECTORIES CAN CHANGE

The specified directory does not exist. Make sure you have entered the correct name and retry.

This Utility program produces formatted program listings on the printer. This Utility can also produce cross reference listings of numeric variables, string variables, and statement numbers, and available variable maps for one or more programs.

To invoke this Utility:

ENTER: *KPSD and 'CR'

This entry may also be entered as *K or as the function number indicated on the BASIC Utilities Menu. The system will call up the *KPSD—PRINT PROGRAM LISTING screen with the following prompt displayed:

SELECT PRINTER: pp pp:

(where: pp is a configured printer.

Printers in use are shown in background mode.)

Enter the name of the printer where the output is to be sent. Enter CR to be send it to 'LP'. If only one printer is configured, this question will not be asked. The system will respond with the following prompt:

PAGE WIDTH (CR = 80)

Enter the maximum number of characters to be printed on one line. This value must be between 55 and 132. The default is 80. The system will respond with the following prompt:

MAXIMUM LINES PER PAGE (CR = 60):

Enter the number of lines to be printed on each page. This number must be between 30 and 256. The default is 60. The system will respond with the following screen:

1) LIST PROGRAMS	: Y
2) CROSS REFERENCE NUMERIC VARIABLES	: N
3) STRING	: N
4) GLOBAL	: N
5) STATEMENT NUMBERS	: N
6) LIST AVAILABLE VARIABLES	: N
7) LIST AVAILABLE VARIABLES—GLOBAL	: N

ENTER SELECTION TO CHANGE OR 'CR':

Enter the numbers of the selections you want to change (they may be stacked, i.e., entering 1234 and 'CR' will change selections 1, 2, 3, and 4). If all selections are 'N' or if selection 4 is 'Y' and selections 2 and 3 are 'N', the program will not accept the 'CR'. The system will respond with the following prompt:

TITLE

Enter a character string to appear at the top of each page of the output listing. If CR is depressed, the title will be taken from the first line of the listed programs if they are REM statements. If they are not REM statements, the Title field will be blank. The system will respond with the following prompt:

DISC ID:

Enter the number of the directory on which the program resides. The system will respond with the following prompt:

ENTER FILENAME OR MASK; (@) FOR ALL, (CR) TO END

Enter up to 48 files names or masks. Entering @ will include all the files. Depress CR to stop entering names and masks. Depress CR again and the Utility will print the listing. When the listing is complete, the system automatically returns the BASIC Utilities Menu.

NOTES

This Utility assumes that the programs are broken into the following sections and it prints appropriate headers: 0001-0099 IDENTIFICATION; 0100-0999 INITIALIZATION; 1000-6999 MAIN PROCESSING; 7000-7999 SUBROUTINES; 8000-8999 ERROR PROCESSING; and 9000-9999 PROGRAM TERMINATION. The start of the subroutine section may be advanced by including a REM with '*S*' as the first three characters. This will cause the subroutine heading to be shown with the range nnnnn-07999 where nnnnn is the statement number of the REM.

Variable Cross Reference Listings show each variable used followed by the numbers of all the statements in which it is refer-

enced. The Global Listing shows each variable followed by the names of the programs in which it is used. The Statement Number Cross Reference shows each statement number followed by the numbers of the statements that reference it. The Available Variables Listing shows all valid BASIC variable names, including numeric arrays, with the variables that have been used replaced with dashes (--).

ERROR MESSAGES

PRINTER IN USE

The requested printer is being used by another task; select another.

PRINTER NOT READY

The requested printer is offline; select another.

LIMIT IS 48 NAMES/MASKS

The Utility will not accept any more file names or masks and will begin its directory search.

CAN'T DEFINE WORK FILE

The work file used in cross referencing could not be created. The Utility returns to the option selector.

ERROR 0 HAS OCCURRED, PRINTER PROBABLY OFFLINE

Check if the printer is offline or out of paper. Enter CR to continue.

***LPSD**

DISPLAY DISK DIRECTORY

This Utility program lists portions of a disk directory on the terminal, one screen at a time. The directory may be sorted alphabetically or by location on the disk. Portions of the directory may be selected by alphabetical range, location range, or by pattern-matching masks. For each file, file name, file type, starting sector, number of sectors, record length, number of records, and key length are shown.

To invoke this Utility:

ENTER: *LPSD and 'CR'

This entry may also be entered as *L or as the function number indicated on the BASIC Utilities Menu. The system will call up the *LPSD—DISPLAY DISK DIRECTORY screen with the following prompt displayed:

DISC ID:

Enter the number of the directory you wish to list. The system will display the name of the selected directory and the following prompt:

AVAILABLE SECTORS, LOCATION, MASKS, OR NAME (A/L/M/N)

Enter A to display the location and length of unused blocks on the disk. Enter L to extract and sort the directory by location, M to select files by masks, or N to select files by name and sort them alphabetically.

The following list outlines the prompts generated by the system in response to selections from the previous prompt:

FROM SECTOR

Enter the location on the disk where you want to begin displaying the directory. Depressing CR sets the value to zero.

TO SECTOR

Enter the number of the sector where the listing is to stop. 'CR' sets this value to the highest numbered sector on the disk.

**TYPES (I = IND, D = DIR
S = SRT, R = SER, P = PGM)**

Enter up to five characters representing the types of files to be displayed. Enter CR to display all file types. If a character other than I, D, S, R, or P is entered, or if more than five characters are entered, the user will be prompted again.

**ENTER FILENAME OR
MASK; (@) FOR ALL,**

Enter up to 48 file names or masks. Entering @ will list all the names in the directory. If a file does not exist on the specified disk, or if it is not of the correct type, the name will be rejected. Depress CR to stop entering names and masks.

FROM NAME

Enter the name of the first file to be displayed. Depressing CR will set the first file name to the first name in alphabetical order.

TO NAME

Enter the name of the last file to be displayed. Depressing CR will set the last file name to the last name in alphabetical order.

When the entries for the applicable prompts have been made, the screen will display the directory information. At the bottom of each screenful, the following message is displayed:

HIT (CR) TO CONTINUE

Depress CR to continue with the listing. When the listing is complete, the following prompt will be displayed at the bottom of the screen:

HIT (CR) FOR NEW LISTING

Depress CR to return to the AVAILABLE SECTORS, LOCATION, MASKS, OR NAME prompt. Select another way of listing the same directory or depress CTL-III (F3) to return to the DISC ID field. You may enter another disk directory to list, or depress CTL-III (F3) or CTL-IV (F4) to return to the BASIC Utilities Menu.

ERROR MESSAGES

INVALID FILE NAME

The requested directory does not exist; enter another.

DISK nn NOT CONFIGURED

The disk number given is invalid or the disk does not have a Thoroughbred/OS directory.

'xxxxxx' NOT A DIRECTORY

The directory named is not a directory. Enter another name.

LIMIT IS 48 NAMES/MASKS

The Utility will not accept any more than 48 file names or masks.

MPSD*DISPLAY TASK ALLOCATION**

This Utility program displays the number of bytes available in each memory bank and the number of bytes used by each task in each bank.

To invoke this Utility:

ENTER: *MPSD and 'CR'

This entry may also be entered as *M or as the function number indicated on the BASIC Utilities Menu. The system will call up the *MPSD—DISPLAY TASK ALLOCATION screen. The following is a sample screen:

***MPSD—DISPLAY TASK ALLOCATION**

BANK 1	BANK 2	BANK 3
28943	57325	59976
1280 T0	4279 T1	3072 T2

CONFIGURED TERMINALS: T0 T1 T2
THIS IS TERMINAL T0

HIT (CR) TO END

Depress CR to end the display of the task allocation. The system will redisplay the BASIC Utilities Menu.

This Utility program is used to configure serial ports, disk directories, mnemonic tables, system options, and memory banks for your system.

To invoke this Utility:

ENTER: *NPSD and 'CR'

This entry may also be entered as *N or as the function number indicated on the BASIC Utilities Menu. The system will call up the *NPSD—BASIC CONFIGURATOR screen. The following is a sample screen:

**** SMC BUSINESS BASIC CONFIGURATOR—VERSION 6—30APR83 ****

1—HARD DISK UNIT 0
2—FLOPPY DISK UNIT 0

SELECT CONFIGURATION SOURCE:

ENTER: appropriate response

Enter 1 to modify configuration of the hard disk. Enter 2 to modify the configuration of the floppy disk. The system will issue the following notification:

READING FROM 'xxxxxx' DISK UNIT 0

(where: 'xxxxxx' is the device selected for configuration)

When the read is complete, the system will call up the following screen:

THE FOLLOWING TYPES OF SYSTEM INFORMATION ARE CONFIGURABLE

1. SERIAL PORTS
2. DISK DIRECTORIES
3. MNEMONIC TABLES
4. SYSTEM OPTIONS
5. MEMORY BANKS

NOTE: ENTER 'X' TO CANCEL CONFIGURATION JOB
'Y' TO END JOB NORMALLY (SAVE NEW CONFIGURATION)

ENTER NUMBER OF CONFIGURATION TYPE:

Enter the number of the system information that you wish to modify. When you have finished the modifications for the type information selected, you will be returned to this screen to either select another type to modify or to end the Utility by cancelling or saving the new configuration. When you enter a Y, the system will check to assure that all parameters have been correctly set. If they are not, the following message will appear:

**INVALID CONFIGURATION!
CHECK ALL DEFINED PARAMETERS.**

You should then re-check all parameters that were defined before you continue. Entering X cancels all changes made and returns you to the BASIC UTILITIES menu. Enter Y and depress CR to retain all the changes. The following command will appear:

RE-BOOT SYSTEM TO USE NEW CONFIGURATION

You must boot the system for the new configuration to be in effect.

1. SERIAL PORTS

Enter 1 on the BASIC CONFIGURATOR menu to modify serial ports. The system responds with a screen, listing the ports configured for the system. The following is a sample screen:

SERIAL PORT CONFIGURATION

PORT	DEVICE	MODEL	BAUD	AUTO- START	BANK	PROGRAM
0	T0	9500	09600	Y	1	**PSD
1	T1	9500	09600	Y	1	
2						
3						

ENTER PORT NUMBER TO MODIFY (C.R. TO EXIT):

Enter either the number of the port that you wish to modify or depress CR to return to the BASIC CONFIGURATOR menu. If you enter the number of a port already configured, the following prompt appears:

DELETE ENTRY? (Y/C.R.) N

This prompt gives you the choice of either deleting or modifying the existing entry. Enter Y and the system will return the SERIAL PORT Menu with the entry deleted.

Enter N or depress CR (default is N) to keep the entry and modify it. The following screen appears if you enter N or if you have selected a port not yet configured:

**ESCAPE KEY WILL CANCEL CHANGES
RETURN KEY WITH NO MORE INPUT WILL GO TO NEXT FIELD**

**DEVICE CODES ARE T0-T9 (TERMINALS); LP AND P1-P9 (PRINTERS)
ENTER DEVICE CODE:**

Enter the device code and depress CR. NOTE: This system currently does not support modification of LP and P1-P9. The following prompt will appear:

ENTER MODEL CODE

Enter the model code of your terminal and depress CR. The following prompt will appear:

ENTER BAUD RATE

Enter the baud rate for the port that you are modifying and depress CR. The system responds with the following prompt:

AUTO START (Y/N) N

This refers to the automatic start-up of the terminals. At least one terminal must be configured for AUTO START. 'N' is the default. Enter Y and depress CR, or depress CR for 'N'. If you enter N, the system will return the SERIAL PORT screen with the new information listed. If you enter Y, the following prompt appears:

ENTER BANK NUMBER

Enter the bank number for which this port will be configured. (To see the memory banks configured for the system, select option 5, "MEMORY BANKS" on the BASIC CONFIGURATOR menu.) Enter the bank number and depress CR. The following prompt appears:

ENTER PROGRAM (BLANK FOR CONSOLE MODE)

Enter the name of the program to be brought up upon the automatic start-up of the terminal and depress CR. If you do not enter a program name, the console mode will automatically be brought up upon signon. The system then returns the SERIAL PORT screen with the new information listed. Make any changes to other ports following the same procedure.

If there are no further changes to be made, depress CR to return to the BASIC CONFIGURATOR menu. Select another type to configure, or enter either X to cancel the configuration, or Y to save the configuration.

2. DISK DIRECTORIES

Enter 2 from the BASIC CONFIGURATOR menu to configure disk directories, and depress CR. The screen responds with a list of the directories configured for the system. The following is a sample screen:

DISK CONFIGURATION

DIRECTORY CODE	TYPE H(ard)/F(loppy)	UNIT NUMBER	DIRECTORY NAME	TYPE OF BUFFERING
D0	H	0	UTIL	READ/WRITE
D1	H	0	PROG	READ/WRITE
D2	H	0	DATA	READ/WRITE
D3	H	0	DOCU	READ/WRITE
D4	H	0	DEMO	READ/WRITE
D5	H	0	SURV	READ/WRITE
D6	H	0	VOL1	READ/WRITE
D7				
D8				
D9	F	0	MBOS	

ENTER DIRECTORY CODE TO MODIFY (C.R. TO EXIT): _____

NOTE: In this sample screen, directories 7 and 8 are not configured. Therefore, only 8 directories can be accessed at any one time. Enter the directory code that you wish to change. If the directory is already configured, the following prompt appears:

DELETE ENTRY? (Y/C.R.) N

Enter Y to delete the directory entry. The system will return the DISK CONFIGURATION screen with the entry deleted.

Enter N or depress CR (default is N) to retain the entry and modify selected information.

The following prompt appears in response to 'N' or for a selection of a directory not yet configured:

ENTER 'H' FOR HARD DISK, 'F' FOR FLOPPY:

Enter the device on which the directory you are configuring resides. The device type configured for the existing directory (if you are modifying an existing directory) is the default. 'H' is the default for directories not configured. The system then responds with the following prompt:

ENTER DRIVE NUMBER (0-3):

Enter the drive number for the directory you are configuring. The drive number configured for the existing directory is the default. If the directory is not configured, '0' is the default. The system responds with the following:

ENTER DIRECTORY NAME: _____

Enter a 1-4 character name for the directory you want to configure. The system responds with the following prompt:

**ENTER CODE FOR TYPE OF DEVICE BUFFERING
(N = NONE, R = READ-ONLY, W = READ AND WRITE): _____**

This prompt refers to the buffering specification for each device. Enter N to transfer to and from the device immediately. Enter R to have read-only data held in the buffer. Enter W to have all data, whether being written or read held in the buffer.

NOTE: It is absolutely essential to assure that a "sync" occurs (a sync updates sector cacheing buffers to disk *only* when READ/ WRITE is in effect, regardless of block size). When either changing a floppy, re-booting, or turning the machine off, to assure the occurrence of a sync, and reflecting the particular process you are involved in, you should either:

- a. disable the disk
- b. get into the console mode (by depressing ESCAPE or F4)
- c. let machine sit idle for x seconds (the time will depend upon set parameters)

3. MNEMONIC TABLES

Enter 3 from the BASIC CONFIGURATOR menu to configure mnemonic tables. The system responds with a screen listing model tables available. The following is a sample screen:

NOTE: These tables are preset and should not require modification. If, however, a new terminal not recognized by Thoroughbred/OS is added to your system, this function will have to be performed. Please consult your terminal manual for the proper settings and a definition of the mnemonic codes.

THE FOLLOWING MNEMONIC TABLES ARE AVAILABLE FOR MODIFICATION:

0300	0950	1650	7220	7250	8000	8001	8019	9500	9900
------	------	------	------	------	------	------	------	------	------

SELECT A TABLE TO MODIFY (C.R. = EXIT TO MAIN MENU)

Enter one of the tables. If you are defining a new terminal to the system, it is advantageous to select the table that is most similar to the one to be configured. The system responds with the following:

**ENTER NEW MODEL NUMBER
(C.R. = MODIFY SELECTED TABLE)**

Enter a model number, or depress CR to modify the selected table. The system responds with the following prompt:

**POSITION CURSOR SEQUENCE
COLUMN OR LINE FIRST (C OR L)**

Enter the correct response according to your terminal manual. Enter C to position the cursor according to column first, line second (e.g. 1,20 refers to column 1 line 20). Enter L to position the cursor with line first, column second (e.g. 1,20 refers to line 1, column 20).

The system responds with the following prompt:

**ENTER SECOND CHARACTER OF ESCAPE SEQUENCE
(C.R. = ____)**

Enter a hexadecimal value for the second character of the escape sequence in accordance with your terminal manual. Depress CR to retain the value of the selected model table as displayed. The system responds with the following prompt:

ENTER LINE CODE (C.R. = ____)

Enter a value to represent the setting of the uppermost line of the screen in accordance with your terminal manual. NOTE: This is set in conjunction with the prompt below to position the cursor in the upper lefthand corner. Depress CR to retain the value of the selected model table as displayed. The system responds with the following prompt:

ENTER COLUMN CODE (C.R. = ____)

Enter a value to represent the setting of the upper left column in accordance with your terminal manual. Depress CR to retain the value of the selected model table as displayed. The system responds with the following prompt:

ENTER CODE FOR Fn (C.R. = nn)

This prompt will be generated for each defined function key. Enter a new value in accordance with your terminal manual or depress CR to retain the value of the selected model table as displayed. When the final entry of the series has been entered, the system responds with the following prompt:

**ENTER I TO IGNORE MNEMONIC OR
ENTER E TO CAUSE ERROR = 29 OR
RETURN TO KEEP OLD VALUE OR
ENTER NEW VALUE AND RETURN**

FOR MNEMONIC CODE 'xx'

OLD VALUE: (xxxxxx)

NEW VALUE:

This prompt will be displayed for each mnemonic configured. Enter a new value; enter I to have the system ignore the mnemonic when it is encountered; enter E to have the system generate an Error 29 when it is encountered; or depress CR to retain the value as displayed in 'OLD VALUE'.

Please refer to your terminal manual for a listing of the applicable mnemonics and appropriate values.

When the defined mnemonics have all been listed, the system responds with the following prompt:

ENTER NEW MNEMONIC CODE (C.R. IF NONE):
ENTER NEW VALUE

Enter any desired mnemonic not previously defined. Depress CR when you have finished; the system returns to the Configurator Menu.

4. SYSTEM OPTIONS

This field allows you to change selected system options. Enter 4 and depress CR to select this option. The system responds with the following screen:

SYSTEMS OPTIONS

1—PARALLEL PRINTER MODEL CODE	1650
2—MAXIMUM NUMBER OF GHOST TASKS	2
3—SIZE OF INTERNAL FILE DICTIONARY	72
4—DATE DELIMITER	/
5—DATE FORMAT	M
6—EDIT DELIMITERS	□
7—RESIDENT COMPILER AND LISTER	Y
8—DEVICE BUFFERING SECTOR MULTIPLE	1
9—DEVICE BUFFERING SYNC INTERVAL	10

ENTER OPTION NUMBER TO MODIFY (C.R. TO EXIT):

OPTIONS

1. Parallel Printer Model Code

This option refers to the printer model configured for your system.

Enter 1 and depress CR to modify this option. The system responds with the following prompt:

SELECT PARALLEL PRINTER MODEL 0300 OR 1650:

Enter the model number of the printer you want to configure for your system. The model for which the system is currently configured is the default value. Enter the model number and depress CR. The system will return the SYSTEMS OPTION screen with the new information inserted. Enter the number of another option to be modified or depress CR to return to the BASIC CONFIGURATOR menu.

For some systems the first option refers to whether or not your system has a tape drive. Enter 1 and depress CR to modify this option. The system responds with the following prompt:

WILL THIS SYSTEM INCLUDE A TAPE DRIVE? (Y/N):

Enter Y if your system has a tape drive. Enter N if it does not. The system will return the SYSTEMS OPTIONS screen with the new value inserted. Enter the number of another option to be modified or depress CR to return to the BASIC CONFIGURATOR menu.

2. Maximum Number of Ghost Tasks

This option allows you to modify the number of Ghost Tasks configured for your system. Enter 2 and depress CR to modify this option. The system responds with the following prompt:

ENTER MAXIMUM NUMBER OF GHOST TASKS (0-4):

Up to 4 Ghost Tasks can be configured for the system. The number of Ghost Tasks currently configured for the system is the default value. Enter the number you wish to have and depress CR. The system will return the SYSTEMS OPTIONS screen with the new number inserted. Enter the number of another option to be modified or depress CR to return to the BASIC CONFIGURATOR menu.

3. Size of Internal File Dictionary

This option allows you to modify the size of the internal file dictionary. Enter 3 and depress CR to modify this option. The system responds with the following prompt:

**THE MAXIMUM REQUIRED DICTIONARY SIZE IS n1
ENTER THE DESIRED INTERNAL DICTIONARY SIZE (n2-255):**

The system calculates n1 and n2. Enter the desired size, from n2 to 255. The current value is the default. The maximum required size is n1. The system will return the SYSTEMS OPTIONS screen with the new value inserted. Enter the number of another option to be modified or depress CR to return to the BASIC CONFIGURATOR menu.

4. Date Delimiter

This option allows you to modify the date delimiters used by the system. Enter 4 and depress CR to modify this option. The following prompt will appear:

**ENTER DEFAULT DATE DELIMITER
(OTHER THAN SPACE OR COMMA):**

Enter any character except a space or comma and depress CR. The system will return the SYSTEMS OPTIONS screen with the new date delimiter inserted. Enter the number of another option to be modified or depress CR to return to the BASIC CONFIGURATOR menu.

5. Date Format

This option allows you to modify the format of the date. Enter 5 and depress CR to modify this option. The system responds with the following prompt:

ENTER DATE FORMAT (M = MDY,D = DMY,Y = YMD):

Enter either M, D, or Y and depress CR. The system will return the SYSTEMS OPTIONS screen with the new date format inserted. Enter the number of another option to be modified or depress CR to return to the BASIC CONFIGURATOR screen.

6. Edit Delimiters

This option allows you to modify the characters used in editing BASIC line items in console mode. Enter 6 and depress CR to modify this option. The system responds with the following prompt:

ENTER BEGIN AND END DELIMITERS FOR EDIT DIRECTIVE:

Enter the delimiters you want to use and depress CR. The system will return the SYSTEMS OPTIONS screen with the new edit delimiters inserted.

7. Resident Compiler and Lister

This option allows you to make the Compiler and Lister either resident or transient. Enter 7 and depress CR to modify this option. The following prompt will appear:

MAKE COMPILER AND LISTER MEMORY RESIDENT (Y/N):

The default is the current value of this option. Enter the desired answer and depress CR. The system will return the SYSTEMS OPTIONS screen with the value inserted.

8. Device Buffering Sector Multiple

This option allows you to select the blocking factor (i.e., the number of sectors read as a group) used on the diskette. Enter 8 and depress CR to modify this option. The system responds with the following prompt:

SELECT 1, 2, OR 4 SECTORS PER DEVICE BUFFER [n]

Enter the number of sectors you wish to use as the blocking factor. Depress CR to retain the existing value. The system will return the SYSTEMS OPTIONS screen with the value inserted.

9. Device Buffering Sync Interval

This option allows you to modify the length of time data is held in a buffer before being written to the disk. Enter 9 and depress CR to modify this option. The system responds with the following prompt:

ENTER DEVICE BUFFERING SYNC TIME (5 TO 30 SECONDS)

Enter the amount of seconds you want the data to be held and depress CR to retain the existing value. The system returns the SYSTEMS OPTIONS screen with the new value inserted. (Refer back to note in device buffering section, page 9-55, for related information.)

5. MEMORY BANKS

Enter 5 on the BASIC CONFIGURATOR menu to modify memory banks configured for the system, and depress CR. The following is a sample of the screen that appears:

MEMORY CONFIGURATION

---AREAS--FOR---

LINE NUMBER	ADDRESS RANGE	BANK NUMBER	MAX # TASKS	PROGRAMS AND DATA	RECORD BUFFERS	SYSTEM TABLE	DEVICE BUFFER
	0K-64K			*** BASIC OPERATING SYSTEM ***			
	64K-79K			*** OPERATING SYSTEM TABLES ***			
	79K-96K			*** COMPILER AND LISTER ***			
1	96K-128K	1	2	53,690	10,240	1,606	26K
2	128K-192K	2	2	58,922	5,008		1,606

ENTER LINE NUMBER TO MODIFY (C.R. TO EXIT): _____

Enter the desired line number.

ENTER MAXIMUM NUMBER OF TASKS IN THIS BANK:

Enter the number of tasks the bank will be configured for.

MINIMUM SIZE FOR RECORD BUFFER AREA IS: ENTER SIZE FOR RECORD BUFFER AREA:

Enter the size of the record buffer area for this bank. The minimum acceptable value is 784/per task. The default value is the value currently assigned. The value specified should be greater than the size of the largest record. The system responds with the following prompt:

REMAINING AVAILABLE AREA IS: nnnnnn
ENTER SIZE OF ANY DEVICE BUFFER AREA:

Enter a value to represent the size for any device buffer. If a size is entered that is more than the remaining available area, this feature will not work. When a value has been entered, the system returns the MEMORY BANKS screen with the new value inserted. Select another bank to define or depress CR to return to the BASIC CONFIGURATOR screen. Select another type to configure, or enter either X to cancel your changes and retain the old configuration or enter Y to save the new configuration. The system automatically returns the SMC BASIC Utilities Menu.

NOTES

This Utility should be run as an independent task. An error will occur if any files are open for any task on the system when this utility is run.

***OPSD**

DEFINE FILES

This Utility program creates and allocates space for new programs and files.

To invoke this Utility:

ENTER: *OPSD and 'CR'

This entry may also be entered as *O or as the function number indicated on the BASIC Utilities Menu. The system will call up the *OPSD—DEFINE FILES screen with the following prompt displayed:

ENTER FILE TYPE (I = IND, D = DIR, S = SRT, R = SER, P = PGM)

Enter the type file you are creating. The following list outlines the prompts generated by the system for the various file types:

- | | |
|--------------------------|--|
| KEY SIZE | Sort and direct files only. Enter the length of the file key. The range is 2 to 56. |
| RECORD SIZE | Enter the record length, in bytes. For indexed, serial, and sort files only. |
| NUMBER OF PAGES | For Program files only. Enter the number of 256-byte pages needed. |
| NUMBER OF RECORDS | Enter the number of records needed. For sort files, enter the number of keys needed. |

When the applicable entries have been made, the system will respond with the following prompt:

DISC NUMBER

Enter the number of the disk where the file is to reside. The system will respond with the following prompt:

FILE NAME

Enter the name of the file to be created. The system will respond with the following prompt:

SECTOR

Enter the number of the sector where the file is to be defined. A 'CR' will cause the file to be defined at the first available location on the disk.

When the file has been created, the system responds with the following message:

FILE XXXXXX DEFINED AT SECTOR YYYY (HIT CR)

Depress CR and the system will return the ENTER FILE TYPE prompt. Depress CR again and the system redisplay the SMC BASIC Utilities Menu.

ERROR MESSAGES

INVALID KEY SIZE. RANGE 2 TO 56

This message indicates that a key size was entered out of the allowable range. Enter a value between 2 and 56.

INVALID RECORD SIZE. RANGE 1 TO 32767

This message indicates that a record size was entered out of the allowable range. Enter a value from 1 to 32767.

INVALID NUMBER OF PAGES. RANGE 1 TO 127

This message indicates that a value for number of pages was entered that is out of the allowable range. Enter a value from 1 to 127.

INVALID NUMBER OF RECORDS. RANGE 1 TO 8388607

This message indicates that an invalid value was entered for the number of records. Enter a value from 1 to 8388607.

INVALID DISK NUMBER

This message indicates that a number was entered of a disk that either does not exist or is not configured. Enter a valid number.

FILE ALREADY EXISTS ON DISK xx.

This message indicates a file with the same name already exists on the specified disk. Enter another name.

ILLEGAL FILE NAME

This message indicates that the name given is longer than 8 characters or contains a non-alphanumeric character other than '*'. Enter another file name.

INVALID SECTOR. RANGE 0 TO XXXX

(where: xxxx is highest sector on disk)

This message indicates the sector value entered is out of range. Enter a valid sector number.

FILE ALREADY DEFINED

This message indicates that a file with this name already exists. Enter a new file name.

NOT ENOUGH ROOM ON DISK

This message indicates that the file could not be created because a block large enough to contain it could not be found on the requested disk.

SECTORS xxxx TO yyyy NOT AVAILABLE

This message indicates the file could not be created because all or part of the requested sectors on the disk have already been allocated to another file. Enter another sector number.

INVALID PARAMETER IN DEFINE STATEMENT

This message indicates that the file was not created because an invalid entry was made and went undetected.

ERROR xx OCCURRED DEFINING FILE

This message indicates an unexpected error occurred while trying to define the file.

This Utility program rennumbers program statements according to start, stop and increment parameters supplied by the user. Programs may be renumbered in sections, but the order of statements may not be changed. All reference numbers in the program are correctly updated.

To invoke this Utility:

ENTER: *PPSD and 'CR'

This entry may also be entered as *P or as the function number indicated on the BASIC Utilities Menu. The system will call up the *PPSD—RENUMBER PROGRAMS screen with the following prompt displayed:

DISC ID

Enter the number of the disk on which the program resides. The system will respond with the following prompt:

ENTER FILENAME OR MASK; (@) FOR ALL, (CR) TO END

Enter up to 48 file names or masks. @ will include all names in the directory. Depress CR to stop entering names and masks. Depress CR again and the system will respond with the following prompt:

ENTER RENUMBER PARMS START 0001 END

For the first block, the first statement number is 0001; for remaining blocks, it is one greater than the end of the previous block. Enter the number of the last statement in this block. Depressing CR sets this entry to 9999. The system will respond with the following prompt:

AS

Enter the statement number that the first statement in the block is to be renumbered to. Depressing CR sets this entry to the same as the start of the block. The system will respond with the following:

STEP

Enter the increment to be used for this block. Depressing CR sets this entry to 0010. If the entry is 0, this portion of the program will not be renumbered.

The Utility will prompt for renumber parameters until it has obtained the STEP option for the block ending at line 9999.

The system will issue a RENUMBERING message along with the statement numbers it is renumbering. When the procedure is complete, the system automatically returns the SMC BASIC Utilities Menu.

NOTES

If a program is uninitialized or blank, the message PROGRAM SKIPPED will appear and the Utility will proceed to renumber the next program. While renumbering, the Utility displays the name of the program it is working on, and the old and new line numbers on the line it is currently renumbering.

ERROR MESSAGES

LIMIT IS 48 NAMES/MASKS

The Utility will not accept any more file names or masks.

MUST BE NUMERIC

Enter a valid statement number or numeric.

MUST BE A POSITIVE INTEGER

Enter a valid statement or increment.

LINE NUMBER OVERFLOW!! PROGRAM NOT RENUMBERED

Renumbering the program would result in a line number greater than 65534. The Utility continues with the next program.

LINE NUMBER ERROR!! NEW LINE LESS THAN PREVIOUS

Renumbering the program would have resulted in the first line of a block being numbered lower than the last line in a previous block. The Utility continues with the next program.

CAN'T DEFINE WORK PROGRAM!! ERROR = xx

The Utility could not define a scratch file due to space limitations. It attempts to proceed to the next program.

***QPSD**

STRING SEARCH AND REPLACE

This Utility program searches programs for specified strings and, optionally, replaces the strings with user-specified strings.

To invoke this Utility:

ENTER: *QPSD and 'CR'

This entry may also be entered as *Q or as the function number indicated on the BASIC Utilities Menu. The system will call up the *QPSD—STRING SEARCH AND REPLACE screen with the following prompt displayed:

DISC ID:

Enter the number of the disk on which the program(s) resides. The system will respond with the following prompt:

ENTER FILENAME OR MASK: (@) FOR ALL, (CR) TO END

Enter up to 48 file names or masks. @ will include all the programs in the specified directory. Depress CR to stop entering file names and masks. The system will respond with the following prompt:

CTL-1 TO SEARCH ONLY, CTL-2 TO REPLACE

Depress CTL-1 (F1) to display occurrences of the string, without replacing it. Depress CTL-2 (F2) to replace the string. If the string is to be replaced, the system responds with the following prompt:

VERIFY (Y/N)

Enter Y to have the Utility ask before replacing strings, or enter N to have the Utility automatically replace the strings. The system will then respond with the following prompt:

HARD COPY (Y/N)

Enter Y to have all statements containing a search string output to a printer or enter N to output to terminal only. If Y is entered, the system responds with the following prompt:

SELECT PRINTER: pp pp

(where: pp is the configured printer and printers in use are shown in background mode.)

Enter the name of the printer where output is to be sent. Depress CR to send the output to 'LP'. If the requested printer cannot be opened, the user will be asked to request another. If only one printer is configured this question will not be asked.

The system will respond with the following prompt:

START LINE

Enter the number of the statement at which the program is to begin its search. Depress CR to start the search at the first statement in the program. The system will respond with the following prompt:

END LINE

Enter the number of the statement at which the Utility should cease searching. Depress CR to search through to the last line in the program. The system will respond with the following:

SEARCH FOR

Enter the search string. Any number of strings will be accepted. Depress CR to begin the search. If the REPLACE option was selected, the system will respond with the following prompt:

REPLACE WITH

Enter the string to be substituted for the search string previously entered. If the VERIFY option was chosen, the system will respond with the following:

REPLACE WITH ssssss (Y/N)?

(where: ssssss is the replacement string)

Enter Y to replace the search string with the replacement string. Enter N to bypass the replacement and continue with the search. The system will issue the following message as each string is found:

CR TO CONTINUE, CTL-4 TO END

Depress CR to continue with the search. Depress CTL-4 (F4) to end the search and replace function. The system will redisplay the DISC ID prompt. Depress CTL-4 (F4) to return to the BASIC Utilities Menu.

ERROR MESSAGES

SKIPPED DUE TO COMPILATION ERROR

When a replacement was made, the resulting statement contained a syntax error. The Utility program continues searching with the next statement.

ERROR xx EXPANDING PROGRAM, NOT MODIFIED

The Utility was unable to create a larger file to accommodate the expanded program. The Utility continues its search with the next program.

CAN'T EXPAND OR REDEFINE PROGRAM, WAS AT SECTOR xx

The Utility could not create a larger file and was also unable to redefine the original program. The sector number is given so that the user may recover the program. The Utility continues with the next program.

LIMIT IS 48 NAMES/MASKS

The Utility will not accept more than 48 names or masks.

INSUFFICIENT SPACE FOR WORK FILE ON DISK xx.

There is not enough space available on the disk to create a scratch file to copy the program into. The run is aborted.

SPSD*TRANSFER PROGRAMS/FILES**

This Utility program copies groups of files and/or programs from one disk to another.

To invoke this Utility:

ENTER: *SPSD and 'CR'

This entry may also be entered as *S or as the function number indicated on the BASIC Utilities Menu. The system will call up the *SPSD—TRANSFER PROGRAMS/FILES screen with the following prompt displayed:

FROM DISC #

Enter the number of the disk on which the files currently reside. The system will respond with the following prompt:

TO DISC #

Enter the number of the disk to which the files are to be copied. The system will respond with the following prompt:

TYPE: PROGRAM (P), FILES (F), OR ALL (A)

Enter P to copy program files only, F to copy data files only, or A to copy all the files. The system will respond with the following prompt:

ASK (A), REPLACE (R) OR SKIP (S) EXISTING FILES

If a file with the same name as the one being copied already exists in the destination disk, enter R to automatically replace it, enter S to automatically skip it, or enter A to have the system ask if it should be replaced before taking any action.

The system will respond with the following prompt:

ENTER FILE NAME OR MASK; (@) FOR ALL, (CR) TO END

Enter up to 48 file names or masks. Enter @ to transfer all the files. Depress CR to stop entering names. Depress CR again to begin the procedure. The system will display the names of the files being transferred.

When the system has completed the transfer of files, the following prompt appears:

xxx FILES TRANSFERRED, HIT (CR) TO CONTINUE

Depress CR to move another group of files or depress CTL-4 (F4) to end the Utility.

NOTES

This Utility transfers files by creating a file on the destination disk with all the characteristics of the source file and copying from the source file into that file. It then disables the source disk and renames the scratch file. At the end of the Utility, all disks are locally disabled except the source and destination disks.

ERROR MESSAGES

LIMIT IS 48 NAMES/MASKS

The Utility will not accept any more file names or masks and will begin its search.

nnnnnn CAN'T CREATE COPY, NOT MOVED

The Utility could not create the file on the destination disk. It proceeds to the next file.

nnnnnn CAN'T BE OPENED, NOT MOVED

The Utility could not open the specified file and proceeds to copy the next file.

ERROR xx HAS OCCURRED, (CR) TO RETRY

An unexpected error has occurred, probably I/O related.

TPSD*FILE TRANSFER AND EXPAND**

This Utility program copies any program or file and allows changes to any of the following at the same time: disk, sector, key size, record size, number of records, or file name.

To invoke this Utility:

ENTER: *TPSD and 'CR'

This entry may also be entered as *T or as the function number indicated on the BASIC Utilities Menu. The system will call up the *TPSD—FILE TRANSFER AND EXPAND screen with the following prompt displayed:

**FILE NAME:
('CR' TO END)**

Enter the name of the file to be copied, or depress CR to terminate the Utility. When a file name is entered, the system calls up a screen listing file information. The following is a sample file and screen:

**FILE NAME: CUTINT
('CR' TO END)**

*** O-L-D ***

*** N-E-W ***

KEY SIZE	0
RECORD SIZE	256
TOTAL RECORDS	34
DISC NUMBER	5
STARTING SECTOR	2,342

KEY AREA	0
DATA AREA	34
TOTAL SECTORS	34

ENTER NEW DISC

Enter the number of the disk to copy the file to. Depress CR to keep it on the same disk.

The following list outlines the prompts generated by the system for this Utility:

ENTER NEW KEY LENGTH For Sort and Direct files, enter the new key length in bytes or depress CR to keep the same key size. The allowable range is 2 to 56 bytes.

ENTER TOTAL RECORDS REQUIRED Enter the number of records desired or depress CR to keep the same number. For Program files, enter the number of 256-byte pages needed; for Sort files, this is the number of keys.

ENTER SECTOR (CR = ANYWHERE) Enter the number of the sector where the file is to be defined or depress CR to define it at the first available location.

NEW NAME OR (CR) TO REPLACE Enter a new file name or CR to erase the old file and use the same name on the new file.

When the applicable entries have been entered, depress CR. The system will begin transferring the files. When the system has completed the transfer, the following message appears:

xxxxxx HAS BEEN REPLACED (HIT CR)
(where: xxxxxx is the file)

Depress CR and the system redisplay the FILE NAME prompt. Enter another file name or depress CR to end the Utility.

ERROR MESSAGES

WARNING: KEYS WILL BE TRUNCATED

If the new key size is smaller than the old key size, this message is displayed. Depress CR to continue or CTL-III (F3) to enter a new key size.

WARNING: FILE MAY BE TRUNCATED

For indexed files, this is displayed if the number of records desired is less than the current file size. For sort and direct files, it is

displayed if the number of records requested is less than the number of keys in use. Depress CR to continue or CTL-III (F3) to enter a new number of records.

WARNING: PROGRAMS MAY BE SHORTENED

The number of pages requested is not large enough to contain the program. Enter a greater number of pages.

nnnnnn ALREADY EXISTS ON DISK xx

The Utility cannot create the file using this name, enter another name.

NO ROOM ON DISK OR SPECIFIED SECTOR AVAILABLE

The Utility cannot create the file and the execution is aborted.

kkkkkk, TRUNCATION RESULTED IN DUPLICATE KEY CONTINUE (Y/N)?

Shortening a key to the new length resulted in its becoming the same as another key. Enter Y to proceed to the next record or N to stop copying the file.

**ERROR xx HAS OCCURRED.
'CR' TO CONTINUE OR CTL-IV TO END**

An unexpected error has occurred, probably I/O related.

***VPSD**

DISKETTE BACKUP/RESTORE

This Utility program is used to backup and restore directories on a disk.

This Utility allows you to backup all or some of the directories on a disk. For each directory selected, you can choose either Program files only or data files (including Indexed, Sort, and Direct), or both, to be included in the backup. This Utility also has a feature called the *VBU Maintenance Option, which allows you to backup individual files (of either type), from any of the directories on the disk.

The RESTORE function of this Utility allows you to restore all (COMPLETE) or part (PARTIAL) of a backup disk. You may restore the files to the directory where they were originally located, or to a different one. If you select a partial restore, you can choose to restore individual files or a range of files.

To invoke this Utility:

ENTER: *VPSD and 'CR'

This entry may also be entered as *V or as the function number indicated on the BASIC Utilities Menu. The system will call up the *VPSD—DISKETTE BACKUP/RESTORE screen with the following prompt displayed:

- 1. BACKUP**
- 2. RESTORE**

ENTER SELECTION:

ENTER: appropriate selection and 'CR'

BACKUP

Enter 1 to select Backup and depress CR. The system displays a list of the directories on the disk. The following is a sample BACKUP screen.

BACKUP

VOL NUM	TYPE DISC	VOL NAME	—FILES—		# OF DSKTS	—PROGRAMS—		# OF DSKTS	—TOTALS—	
			NUM	SIZE		NUM	SIZE		FL&PG	BYTES DSKTS
0	HD	UTIL								
1	HD	PROG								
2	HD	DATA								
3	HD	DOCU								
4	HD	DEMO								
5	HD	SURV								
6	HD	VOL1								
7	—	---								
8	—	---								
9	FL	MBOS								
—	—	*VBU								

SYSTEM DATE IS mm/dd/yy

DATE CORRECT (Y/N)?

ENTER: appropriate response

Enter N if the date is incorrect. The system re-prompts for the correct date. When the correct date is entered, the system responds with the following prompt:

**SELECT DISK(S) TO BE BACKED UP: (S) SELECT
(D) DESELECT (CR) NO CHANGE**

The cursor is positioned at the first directory. Enter S beside each directory containing files you want to backup. Each directory selected is highlighted. DESELECT (D) is the default. If you have selected the *VBU Maintenance option, please see the *VBU section following this section.

When you have selected all the directories containing files you want backed up:

DEPRESS: 'CR'

The system responds with the following question:

SELECTIONS CORRECT (Y/N)?

ENTER: appropriate response

Enter N to make a change in your selection. A directory previously selected remains selected unless you enter D beside it. Make any necessary changes. When the selections are correct, enter Y and depress CR. The system responds with the following message:

CALCULATING # OF DISKETTES REQUIRED FOR BACKUP, PLEASE WAIT!

As the system calculates the number of diskettes required to backup each selected directory, information is entered on the screen for those directories selected. The following is a sample screen:

BACKUP

VOL NUM	TYPE DISC	VOL NAME	—FILES—		# OF DSKTS	—PROGRAMS—		# OF DSKTS	—TOTALS—		
			NUM	SIZE		NUM	SIZE		FL&PG	BYTES	DSKTS
0	HD	UTIL									
1	HD	PROG									
S2	HD	DATA	12	18K	.03	0	0K	.00	12	18K	.03
3	HD	DOCU									
4	HD	DEMO									
S5	HD	SURV	4	147K	.23	6	24K	.04	10	171K	.26
6	HD	VOL1									
7	—	---									
8	—	---									
9	FD	MBOS									
—	—	*VBU									

SELECT: (F) FILES (P) PROGRAMS (B) BOTH (CR) NO CHANGE

When the system has completed filling in the information, the cursor is positioned at the first directory selected for backup. Select the file type(s) for each directory that you are backing up. Enter F to include only data files, enter P to include only Program files, or enter B to include both file types. The cursor will skip only to those directories previously selected for backup. Depress CR after each entry. When the file types have been selected for each directory included in the backup, the system will fill in the totals for each directory. The following is a sample screen:

BACKUP

VOL NUM	TYPE DISC	VOL NAME	—FILES—		# OF DSKTS	—PROGRAMS—		# OF DSKTS	—TOTALS—		
			NUM	SIZE		FLOPPY	DIR	OVHD:	FL&PG	BYTES	DSKTS
0	HD	UTIL							---	---	---
1	HD	PROG							---	---	---
F2	HD	DATA	12	18K	.03	0	0K	.00	12	18K	.03
3	HD	DOCU							---	---	---
4	HD	DEMO							---	---	---
P5	HD	SURV	4	147K	.23	6	24K	.04	6	24K	.04
6	HD	VOL1							---	---	---
7	—	---							---	---	---
8	—	---							---	---	---
9	FD	MBOS							---	---	---
—	—	*VBU							---	---	---
TOTALS:									23	43K	.08

SELECTIONS CORRECT (Y/N)?

ENTER: appropriate response

Enter N if you want to change any of the file types selected for the directories. The cursor will be positioned at the first selected directory. The file type previously entered remains selected unless you enter another file type in its place. Enter Y when all the selections are correct. The system responds with a message telling you how many diskettes are required for the backup and then issues the following command:

**PLEASE MOUNT BACKUP DISKETTE VOL-01 IN DRIVE 9,
'CR' TO CONTINUE**

Load the diskette into the drive and depress CR. The system responds with the following message:

**LABELING FLOPPY DISK BACKUP VOL-01
DO YOU WANT TO VERIFY FLOPPY (Y/N)?**

ENTER: appropriate response

Enter N to bypass verification. Enter Y to have the system check the floppy for proper format and correct labelling. If more than one diskette is required for the backup, the system will instruct you to insert each diskette as required. The system labels and, if

instructed, verifies each diskette. When all the diskettes required for the backup have been labelled, the system supplies the following table at the bottom of the screen:

H F FILE V V NAME	TYPE FILE	TOTAL BLOCKS	BLOCKS COPIED	UNUSED BLOCKS	COPYING FLOPPY BLOCK
----------------------	--------------	-----------------	------------------	------------------	-------------------------

and issues the following prompt:

MOUNT FLOPPY VOLUME 1, CR TO CONTINUE

Insert the first diskette and depress CR to invoke copying. As each file is copied, it is listed in the table. The following is a sample:

H F FILE V V NAME	TYPE FILE	TOTAL BLOCKS	BLOCKS COPIES	UNUSED BLOCKS	COPYING FLOPPY BLOCK
2 1 CARAJ	DIR	7	3	4	3
2 1 CCRIR	DIR	24	6	18	6
2 1 CCSMS	DIR	8	5	2	6

If more than one diskette is required, the system will let you know when to insert the next diskette. Insert the diskettes in the same order as they were labelled. When the backup is complete, the system responds:

BACKUP SUCCESSFULLY COMPLETED!, 'CR' TO CONTINUE

Depress CR to return to the BASIC Utilities Menu.

***VBU MAINTENANCE OPTION**

The *VBU Maintenance Option allows you to backup individual files. The names of the files selected are maintained in a file called (*VBU). Additional file names can be added at a later point and existing file names can be deleted. Enter S beside *VBU to select this option and depress CR. The system responds with the following screen:

***VBU MAINTENANCE OPTION-----**
(C) CREATE (L) LIST (A) ADD FILES
(D) DELETE FILES (CR) NO CHANGE

EXPLANATION OF FIELDS

- (C) CREATE** Creates space for the files to be defined.
- (L) LIST** Lists the files already defined and maintained by *VBU.
- (A) ADD FILES** Defines files to *VBU. You will be prompted for the volume number and the file name when you enter A.
- (D) DELETE FILES** Deletes files from *VBU. You will be prompted for the volume number and the file name when you enter D.
- (CR) NO CHANGE** Retains the previously defined *VBU list.

If this option has not been previously selected, a list of the files to be backed up must be defined to *VBU. Enter C and depress CR. The system responds with the following prompt:

SAVE FILE NAMES IN *VBU (Y/N)?
CURRENT ENTRIES = NEW ENTRIES:

Enter N if you are creating a list for the first time (i.e., there are no existing file names to save).

For:

- CURRENT ENTRIES** Enter the approximate number of files you will be backing up. This entry can be larger, but not smaller, than the actual number of files you are entering at this time.
- NEW ENTRIES** The NEW ENTRIES field is used when you want to add more entries at a later date. Enter the sum of the CURRENT ENTRIES field and the number of new entries to be added.

When the required information has been entered, depress CR to return to the initial *VBU prompt. Enter A to add the names of the files to backup. The system responds with the following prompt:

VOLUME NUMBER:**FILE NAME:**

Enter the volume number (directory) where the file resides and depress CR. The cursor is positioned at FILE NAME. Enter the name of the file to be backed up and depress CR. The system responds with the following message:

n-name ADDED TO BACKUP CONTROL FILE

(where: n is the volume number and name is the file name)

The cursor is then positioned at FILE NAME, with the volume number from the previous file name remaining. If the next file is on the same volume, simply enter the file name. To add a file from a different volume, depress CR. The cursor is then positioned at the VOLUME NUMBER field. Depress CR after each entry. Each file entered is listed as follows:

n-filename ADDED TO BACKUP CONTROL FILE

If the filename was previously added to *VBU, the following message appears:

n-filename ALREADY IN BACKUP CONTROL FILE

Depress CR when all the files to be backed up have been entered. Depress CR again to return to the *VBU Selection screen. To end the *VBU option and continue with the backup, depress CR. The system responds with the following message:

**CALCULATING # OF DISKETTES REQUIRED
FOR BACKUP, PLEASE WAIT!**

Continue with the Backup procedure as documented from the above prompt.

RESTORE

Enter 2 and depress CR to select Restore. The system responds with a screen listing the directories that are currently enabled.

NOTE: If you want to restore files to a directory not listed, use the *JPSD Utility to enable access. The following is a sample screen:

RESTORE

0 HD UTIL	1 HD PROG	2 HD DATA	3 HD DOCU	4 HD DEMO
5 HD SURV	6 HD VOL1	7 — —	8 — —	9 FL MBOS

FROM VOL	FILE NAME	FILE TYPE	RECORD SIZE	KEY RECORDS	NUMBER BLOCKS	NUMBER
-------------	--------------	--------------	----------------	----------------	------------------	--------

RESTORE COMPLETE BACKUP (Y/N)?

ENTER: appropriate response

Enter Y for a complete restore—restoration of all the files on the backup disk. Enter N for a Partial Restore—restoration of selected backup files. The system responds with the following command:

**MOUNT VOL 1 OF THE BACKUP
TO BE RESTORED
'CR' TO CONTINUE**

Insert the backup diskette and depress 'CR'. If the backup resides on more than one diskette, each diskette must be inserted in the same order as it was backed up.

NOTE: The above message appears for a Complete Restore. The following message appears for a Partial Restore:

**MOUNT A VALID BACKUP VOLUME
'CR' TO CONTINUE**

Insert the backup diskette containing the files to be restored and depress 'CR'.

For either type Restore, the system fills in the screen with the following information: the first file, last file, and total number of files on the backup diskette; the volume number, the number of volumes on which the backup resides; and the date of the backup.

NOTE: COMPLETE will appear in the upper left corner for a complete restore. PARTIAL will appear for a partial restore.

The following is a sample screen:

COMPLETE RESTORE VOLUME NUMBER: 01 of 01 CREATED DATE: 02/15/83

0 HD UTIL	1 HD WHD3	2 HD IDL3	3 HD DAT3	4 HD DOC3
5 — — —	6 — — —	7 — — —	8 — — —	9 FL MBOS

FROM	FILE	FILE	RECORD	KEY	NUMBER	NUMBER	FIRST FILE: — 2-CARAJ
VOL	NAME	TYPE	SIZE	SIZE	RECORDS	BLOCKS	LAST FILE: — 5-SURVOH
							NMBR FILES: 18

LIST FLOPPY DIRECTORY (Y/N)?

ENTER: appropriate response

Enter Y to list all the files residing on the backup disk. Enter N to have the system skip listing all the files. NOTE: Listing the directory is optional and in no way affects the Restore. The following is a sample list of files:

COMPLETE RESTORE VOLUME NUMBER: 01 of 01 CREATED DATE: 02/15/83

0 HD UTIL	1 HD PROG	2 HD DATA	3 HD DOCU	4 HD DEMO
5 HD SURV	6 HD VOL1	9 FL MBOS		

FROM	FILE	FILE	RECORD	KEY	NUMBER	NUMBER	FIRST FILE: — 2-CARAJ
VOL	NAME	TYPE	SIZE	SIZE	RECORDS	BLOCKS	LAST FILE: — 5-SURVOH
							NMBR FILES: -18

2	CARAJ	DIR	128	8	9	3
2	CCRIR	DIR	122	6	38	6
2	CCSMS	DIR	324	6	4	6
2	CDASL	DIR	128	8	2	3
2	CIMRC	DIR	142	6	11	6
2	CMFSL	DIR	66	3	40	6
2	CNACR	DIR	80	17	4	3
2	COPAR	DIR	146	18	125	29
2	CRCKS	DIR	32	8	5	3
2	CSACM	DIR	61	8	12	5

MORE FILES, CR TO CONTINUE

ENTER: appropriate response

If the listing is more than one screenful, the system generates the above message. Depress CR to continue the listing. When all the files have been listed, the following message appears:

END OF DIRECTORY, 'CR' TO CONTINUE

Depress CR to continue with the Restore. The system responds with the following screen:

COMPLETE RESTORE VOLUME NUMBER: 01 of 01 CREATED DATE: 02/15/83

0 HD UTIL	1 HD PROG	2 HD DATA	3 HD DOCU	4 HD DEMO
5 HD SURV	6 HD VOL1	7 — — —	8 — — —	9 FL MBOS

PUT FILES ON ORIGINAL VOL—(Y/N)?

WHICH VOL THEN?

WRITE OVER SAME NAME FILES—(Y/N)?

ASK IF OVER WRITE OK (Y/N)?

FROM	FILE	FILE	RECORD	KEY	NUMBER	NUMBER	FIRST FILE: — 2-CARAJ
VOL	NAME	TYPE	SIZE	SIZE	RECORDS	BLOCKS	LAST FILE: — 5-SURVOH
							NMBR FILES: 18

EXPLANATION OF FIELDS

**PUT FILES ON
ORIGINAL VOL
(Y/N)**

Indicates whether or not the backed up files are to be restored to the directory where they were originally located.

WHICH VOL THEN?

This question will be asked if you entered N to the preceding question. Enter the number of the directory where the files are to be restored.

**WRITE OVER SAME
NAME FILES (Y/N)**

Indicates whether or not you want backup files to replace identically named files existing on the disk. Enter Y and the files will be replaced with the backup files. Enter N and the files will not be restored.

**ASK IF OVER
WRITE OK (Y/N)**

Enter Y and for each file with a name identical to a file already existing on the disk, the system will ask what action to take.

If you are doing a Partial Restore, the following additional fields will be included:

**RESTORE A
RANGE OF FILES
(Y/N)?**

Allows you to choose a range of files to be restored.

FROM: VFFFFFFF These characters represent the volume number and file name of the first file in the range to be restored.

TO: VFFFFFFF These characters represent the volume number and file name of the last file in the range to be restored.

RESTORE ONE FILE AT A TIME (Y/N) FILE NAME: Allows you to restore individual files from the backup disk.

When the required responses have been entered, depress CR and the system responds with the following question:

PARAMETERS CORRECT (Y/N)?

Enter N if you wish to change some of the parameters. The cursor will be positioned at the first field.

Enter Y when the parameters are correct. The system begins the restore procedure. The system lists each file as it is restored. The following is a sample screen:

COMPLETE RESTORE VOLUME NUMBER: 01 of 01 CREATED DATE: 02/15/83

0 HD UTIL	1 HD PROG	2 HD DATA	3 HD DOCU	4 HD DEMO
5 HD SURV	6 HD VOL1	9 FL MBOS		

PUT FILES ON ORIGINAL VOL—(Y/N)?	WHICH VOL THEN?
WRITE OVER SAME NAME FILES—(Y/N)?	ASK IF OVER WRITE OK (Y/N)?

FROM VOL	FILE NAME	FILE TYPE	RECORD SIZE	KEY SIZE	NUMBER RECORDS	NUMBER BLOCKS	FIRST FILE: — 2-CARAJ LAST FILE: — 5-SURVOH NMBR FILES: 18
2	CARAJ	DIR	128	8	9	3	
2	CCRIR	DIR	122	6	38	6	

If you selected the "ASK IF OVERWRITE OK" parameter for backup files with names identical to files existing on the disk, the system generates the following question when a duplicate is found:

OVERWRITE FILE xxxx (Y/N)?

(where: xxxx is the file name)

Enter Y to restore the backup file. The system responds with the following message:

FILE OVERWRITTEN

Enter N to bypass restoring the backup file and the system generates the following message:

FILE BYPASSED

When the RESTORE procedure is complete, the system responds with the following message:

**RESTORE SUCCESSFULLY COMPLETED!,
CR TO CONTINUE**

Depress CR and the BACKUP/RESTORE screen is returned.
Depress CR again to return to the BASIC Utilities Menu.

NOTES

Due to memory considerations, this Utility program should be run as an independent task.

ERROR MESSAGES

**BACKUP LABEL PROCESS TERMINATED ABNORMALLY,
'CR' TO RETRY**

A label could not be created on the backup disk. Retry the procedure. If the problem re-occurs, the floppy is probably damaged or not formatted.

**BACKUP TERMINATED ABNORMALLY
'CR' TO CONTINUE**

An unexpected system error occurred. Retry the procedure. Check to be sure there are no other users on the system.

**FLOPPY DISK NOT CONFIGURED!
BACKUP/RESTORE ABORTED
'CR' TO CONTINUE**

The floppy disk for the Backup/Restore is not configured. Select a configured floppy disk and retry the procedure.

**NO FILES OR PROGRAMS SELECTED FOR BACKUP
'CR' TO CONTINUE**

This message indicates no files or programs were selected for the backup. Make your selection and continue with the backup.

**NOT ENOUGH MEMORY ALLOCATED TO
TERMINAL TO RUN BACKUP**

This message indicates there is not enough memory available to backup the disk. Erase any unnecessary files or add additional memory. Also, check to be sure there are no other users on the system.

**RESTORE TERMINATED ABNORMALLY,
'CR' TO RETRY**

An unexpected system error has occurred. Retry the procedure.

This Utility program produces a cross reference listing of all I/O statements that reference an I/O channel. A list containing the channel number, program name, directive, line number, and for OPEN statements, the name of the file being opened, is sent to a printer. The Utility has the capability of cross referencing all programs run as overlays to the requested program.

To invoke this Utility:

ENTER: *XPSD and 'CR'

This entry may also be entered as *X or as the selection number indicated on the BASIC Utilities Menu. The system will call up the *XPSD—CHANNEL CROSS REFERENCE screen with the following prompt displayed:

SELECT PRINTER pp pp

(where: 'pp' is a configured printer.

Printers in use are shown in the background mode.)

Enter the name of the printer where the output is to be sent. A carriage return sends the output to 'LP'. If the requested printer cannot be opened, the user will be asked to request another. If only one printer is configured, this question will not be asked.

DISK ID:

Enter a valid disk number. If the directory for the specified disk cannot be opened, the user will be asked to supply another disk number.

ENTER FILE NAME OR MASK; (@) FOR ALL, (CR) TO END

Enter up to 48 file names or masks. Entering @ will copy all names in the directory into '*SRT'. If a file does not exist on the specified disk or if it is not of the correct type, the name will be rejected. Enter a carriage return to stop entering names and masks. Depress F4 to end the Utility program without building '*SRT' (see Notes on masks under the *BPSD utility).

CROSS REFERENCE CALLED PROGRAMS (Y/N)

Enter Y to cross reference programs called as overlays to the program being processed. Enter N to cross reference only the requested program.

NOTES

Directives with variables as channel numbers appear after the entries for channel 7. All RUN statements contained in the program appear at the end of the listing. For RUN and OPEN statements, the name of the file being opened or run appears on the listing; if the name was contained in a variable, the word "VARIABLE" appears.

ERROR MESSAGES

PRINTER IN USE

The requested printer is being used by another task; select another.

PRINTER NOT READY

The requested printer is offline; select another.

LIMIT IS 48 NAMES/MASKS

The Utility will not accept any more than 48 file names or masks, and will begin its directory search.

ERROR xx DEFINING WORK FILE

The Utility program was unable to create a sort file. The run is aborted.

nnnnnn IS INITIALIZED OR BLANK

The specified program is empty. The Utility proceeds to the next program.

nnnnnn WAS REFERENCED IN A RUN STATEMENT BUT DOES NOT EXIST

The Utility could not locate a program that was called as an overlay. It proceeds to cross reference the next program.

**nnnnnn WAS REFERENCED IN A RUN STATEMENT
BUT IS NOT A PROGRAM**

The Utility opened a file that was named in a RUN statement and found that it was not a program. It continues with the next program.

ERROR xx HAS OCCURRED, CR TO RETRY

An unexpected error has occurred, probably I/O related. Retry the procedure.

*YPSD

LIST CONFIGURATION

This Utility program lists the configuration of a specified disk.

To invoke this utility:

ENTER: *YPSD

This entry can also be entered as *Y or as the function number indicated on the Utilities Menu.

The *YPSD LIST CONFIGURATION screen will then be displayed with the following message:

ENTER DISK # (0-9):

ENTER: desired disk number

Enter the disk number for which you want to list the Thoroughbred/OS configuration. The following is a sample screen:

*YPSD—LIST CONFIGURATION

CONFIG ON DRIVE	= 0	SYSTEM DIRECTORY SIZE	= 72	TAPE ON SYSTEM	= N
NUMBER OF BANKS	= 2	DATE DELIMITER	= /	NUMBER OF GHOSTS	= 2
NUMBER OF DEVS	= 15	DATE FORMAT	= M	RESIDENT OPTION	= Y
NUMBER OF TASKS	= 8	EDIT DELIMITERS	= □		

DRIVE	TYPE	DIRECTORY	PRINTER	TYPE	BAUD	TERMINAL	TYPE	BAUD	AUTO START
D0	H	UTIL	LP	1650	00000	T0	9500	09600	XX PSD
D1	H	PROG				T1	9500	09600	XX PSD
D2	H	DATA							
D3	H	DOCU							
D4	H	DEMO							
D5	H	SURV							
D6	H	VOL1							
D9	F	MBOS							

ENTER (F4) TO EXIT / (F2) TO PRINT:

ENTER: F2

Depress CTL-II (F2) to print the configuration screen. Depress CTL-IV (F4) to return to the BASIC Utilities Menu.

EXPLANATION OF FIELDS

CONFIG ON DRIVE	The drive on which Thoroughbred/OS is configured.
NUMBER OF BANKS	The number of memory banks configured on the system.
NUMBER OF DEVCS	The number of devices configured on the system. This includes terminals, printers, logical disks and ghost tasks.
NUMBER OF TASKS	The number of tasks configured for all the memory banks.
SYSTEM DIRECTORY SIZE	The maximum number of files open on all the tasks.
DATE DELIMITER	The delimiters used for the format of the date e.g., mm/dd/yy.
DATE FORMAT	The date format configured for the system: mm/dd/yy = M dd/mm/yy = D mm/dd/yy = Y
EDIT DELIMITERS	The characters defined for use in editing SMC BASIC line items in the console mode.
TAPE ON SYSTEM	Indicates whether or not a tape drive is configured for the system.
NUMBER OF GHOSTS	The number of ghost tasks configured for the system.
RESIDENT OPTION	Indicates whether or not the Compiler and Lister are in resident memory.

The bottom section of the screen lists the directory numbers, disk type and directory name for the selected disk. In addition, printers and terminals configured for the selected disk are listed.

ERROR MESSAGES

COULDN'T ACCESS DISK 'n'

This message indicates that the disk you have specified could not be accessed. Check to make sure the specified disk exists, is configured, and then retry the procedure.



APPENDIX A

ASCII CODES — In Hexadecimal and Decimal (American Standard Code for Information Interchange)

First Digit of Hexadecimal Code \ Second Digit of Hexadecimal Code		For Systems with Bit 8 Set to "0"															
0	0	1	2	3	4	5	6	7									
		128	144	160	176	192	208	224	240								
0	NUL	Null	DLE	SPACE	0	@	P	\	P								
			Data Link Escape		Zero	Commercial At		Accent Grave									
1	SOH	129	145	161	177	193	209	225	241								
			DC1	!	1	A	Q	a	q								
2	STX	130	146	162	178	194	210	226	242								
			DC2	"	2	B	R	b	r								
3	ETX	131	147	163	179	195	211	227	243								
			DC3	#	3	C	S	c	s								
4	EOT	132	148	164	180	196	212	228	244								
			DC4	\$	4	D	T	d	t								
5	ENQ	133	149	165	181	197	213	229	245								
			NAK	%	5	E	U	e	u								
6	ACK	134	150	166	182	198	214	230	246								
			ACK	Percent	6	F	V	f	v								

Q	ACK	SYN	&	Q	F	V	f	V
7	ACKnowledge / 135	Synchronous Idle 23 151	Ampersand 39 167	55 183	71 199	87 215	103 231	119 247
	BEL Bell 8 136	ETB End of Trans- mission Block 24 152	' Apostrophe 40 168	7	G	W	g	W
8	BS Backspace 9 137	CAN Cancel 25 153	(Left Parenthesis 41 169	56 184	72 200	88 216	104 232	120 248
	HT Horizontal Tab 10 138	EM End of Medium 26 154) Right Parenthesis 42 170	8	H	X	h	x
9	LF Line Feed 11 139	SUB Substitute 27 155	* Asterisk 43 171	57 185	73 201	89 217	105 233	121 249
	VT Vertical Tab 12 140	ESC Escape 28 156	+ Plus 44 172	9	I	Y	i	y
A	FF Form Feed 13 141	FS File Separator 29 157	, Comma 45 173	58 186	74 202	90 218	106 234	122 250
	CR Carriage Return 14 142	GS Group Separator 30 158	- Hyphen (Minus) 46 174	:	J	Z	j	z
B	SO Shift Out 15 143	RS Record Separator 31 159	. Period 47 175	59 187	75 203	91 219	107 235	123 251
	SI Shift In	US Unit Separator	/ Slash (slant)	;	K	[k	{
C				Semicolon 60 188	76 204	92 220	108 236	Left Brace 124 252
				< Less Than 61 189	L	\	I	I
D				= Equal 62 190	M]	m	} Right Brace 126 254
				> Greater Than 63 191	N	<	n	~ Tilde 127 255
E				?	O	—	o	DEL Delete

APPENDIX B—VFU LOADING

The VFU may be loaded and utilized through the following mnemonic codes:

'SL'—Start Load

'EL'—End Load

'VT'—Vertical Tab (Slew to Channel 6)

'FF'—Form Feed (Slew to Channel 1) (defaults to 66 'LF's)

'S2'—Slew to Channel 2

'S3'—Slew to Channel 3

'S4'—Slew to Channel 4

'S5'—Slew to Channel 5

'S7'—Slew to Channel 7

'S8'—Slew to Channel 8

EXAMPLE

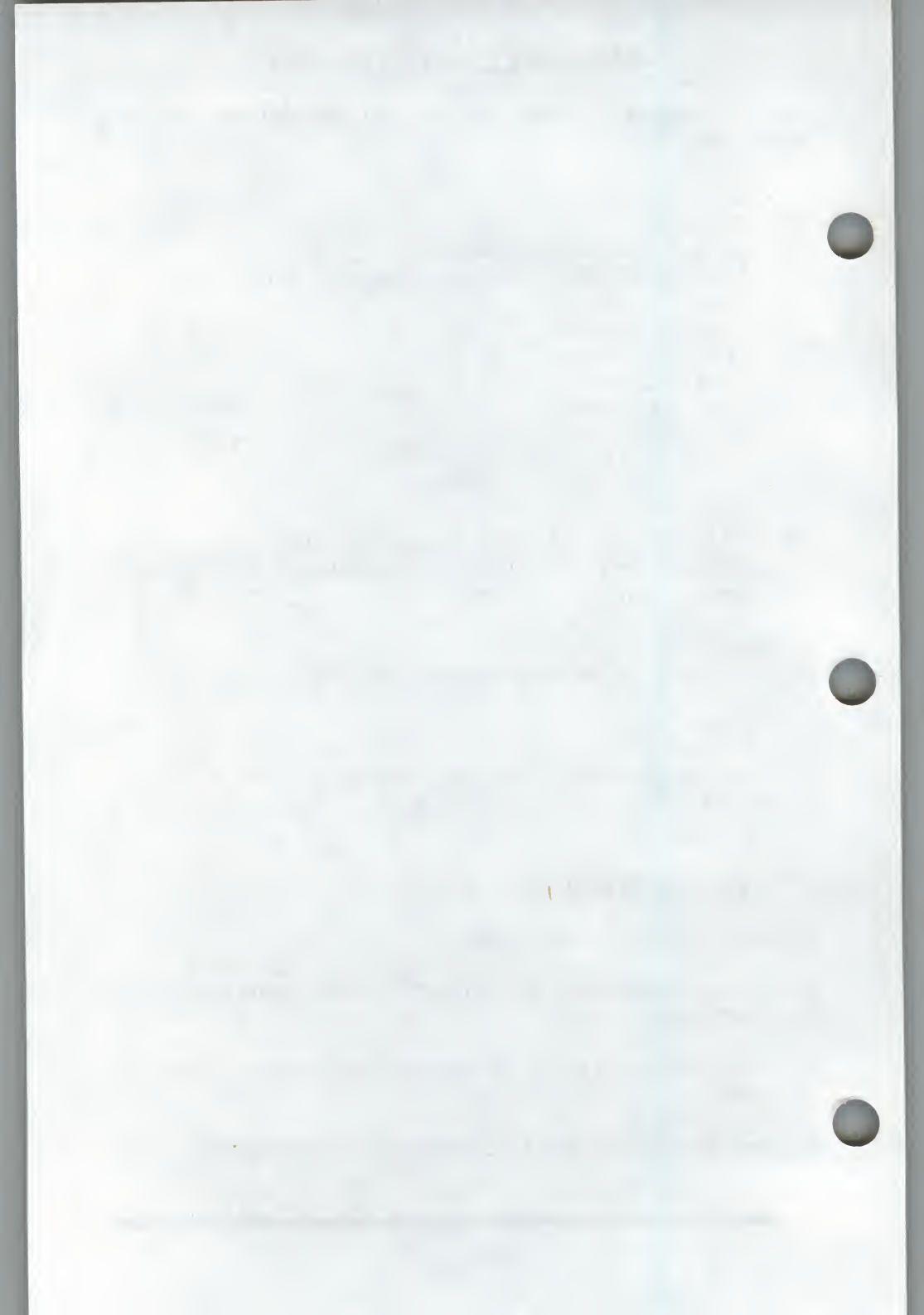
If the VFU is to be loaded with a channel 4 "punch" on line 5, a vertical tab on line 7, a channel 7 on line 12 and a total form length of 20, the following code is necessary:

10 OPEN (1) "LP"

20 PRINT (1) 'SL', "10004060000700000000", 'EL'

Note the following:

1. The string between the 'SL' and 'EL' may only contain numerics. These are translated into the appropriate codes for the printer.
2. A zero is used as a fill character.
3. A six represents vertical tab.
4. A one represents top of form, and there should be only one per VFU load.
5. The length of the string corresponds to the number of lines on a page.
6. Only one channel can be designated per line position.



INDEX

**PSD	9-3
*1PSD	9-5
*2PSD	9-9
*3PSD	9-11
*4PSD	9-13
*APSD	9-17
*BPSD	9-20
*CPSD	9-22
*DPSD	9-25
*EPSD	9-27
*FPSD	9-30
*GPSD	9-32
*HPSD	9-34
*IPSD	9-36
*JPSD	9-39
*KPSD	9-44
*LPSD	9-47
*MPSD	9-50
*NPSD	9-51
*OPSD	9-64
*PPSD	9-68
*QPSD	9-71
*SPSD	9-74
*TPSD	9-76
*VPSD	9-79
*XPSD	9-92
*YPSD	9-95

ABS	2-13
Absolute Value Function	2-13
ADD Directive	5-5
Add Resident Public Program Directive	7-4
ADDR	7-4
Alphanumeric Data	2-4
AND Function	2-56
ASC	2-15
ASCII	1-2, 2-11
ASCII Codes	A-1
ASCII From Hexadecimal Function	2-14
ASCII Function	2-15

ASCII Translation Table	6-48
Assignments	2-11
ATH	2-14
Bank (Memory) Size Function	2-16
BASIC Utilities	9-1
BEGIN Directive	3-4
BIN	2-17
Binary Function	2-17
BSZ	2-16
CALL Directive	7-6
Character Controls	6-9
Character Function	2-18
Character Variables	2-4
CHR	2-18
CLEAR Directive	3-5
CLOSE Directive	5-6
Code Conversion	6-4
Compile Function	2-19
Compound Statements	4-1
Console Mode Processing	3-3
Control/Function Key Input Variable	2-21
CPL	2-19
CRC	2-22
CTL	2-21
Cyclic Redundancy Check	2-22
Data Handling	2-2, 2-5
Data Positioning	6-8
Data Processing	2-5
Data Removal Directives	6-2
Data Representation	2-1
Data Set Description Function	2-23
Data Size Variable	2-25
DAY	2-26
Day (Date) Variable	2-26
DEC	2-27
Decimal Function	2-27
DEF FN	2-28
Define Programmable Function	2-28
Definition Statements	6-2

DELETE Directive	3-6
DIM	2-31, 2-33
Dimension Directives	2-31, 2-33
DIRECT	5-7
Direct File Directive	5-7
Direct Files	5-2
Directives, Special Processing	2-11
DISABLE Directive	5-9
Disk and File Operations	5-1
Disks	5-1
DOM =	6-3
DROP Directive	5-10
DSD	2-23
DSZ	2-25
EDIT Directive	3-8
ENABLE Directive	5-11
END =	6-3
END Directive	3-11
ENDTRACE	3-12
ENTER Directive	7-8
EPT	2-38
ERASE Directive	5-12
ERR	2-35, 2-36
ERR =	6-3
Error Branching	6-3
Error Code Listing	8-3
Error Function	2-35
Error Processing	8-1
Error Processing Capabilities	8-2
Error Task Variable	8-3
Error Trapping	8-2
Error Variable	2-36
Errors, Untrapped	8-2
ESC	6-16
ESCAPE Directive	3-13
Escape Processing	3-3
Exclusive OR Function	2-37
EXECUTE Directive	3-14
Execution Errors	8-1
EXIT Directive	7-9
EXITTO Directive	4-5
Exponent Function	2-38

Expressions	2-5
EXTRACT/EXTRACT RECORD Directive	6-17
FID	2-39
File Accessing	5-4
File Creation	5-3
FILE Directive	5-13
File Identification Function	2-39
File Types	5-2
Files	5-2
FIND/FIND RECORD Directive	6-21
Fixed-Point Data	2-2
FLOATING POINT Directive	2-41
Floating-Point Data	2-2
FOR/NEXT Directive	4-6
FN	2-70
FPT	2-42
Fractional Part Function	2-42
Functions	2-7, 2-28, 2-70
GAP	2-43
Generate Odd Parity Function	2-43
GET Directive	6-25
GOSUB Directive	4-9
GOTO Directive	4-10
Hash Function	2-44
Hexadecimal From ASCII Function	2-45
Highest Disk Sector Function	2-46
HSA	2-46
HSH	2-44
HTA	2-45
IF/THEN/ELSE Directive	4-11
Inclusive OR Function	2-47
IND	2-48
Index Function	2-48
INDEXED	5-15
Indexed File Directive	5-15
Indexed Files	5-2
Input Directives	6-1
INPUT/INPUT RECORD Directive	6-26

Input/Output List Statement	6-31
Input/Output Processing	6-1
Input Verification	6-5
INT	2-49
Integer Format	2-2
Integer Function	2-49
Internal Representation	2-11
IOL =	6-4
IOLIST	6-4, 6-31
IOR	2-47
 KEY Function	 2-50
 LEN	 2-51
LEN =	6-4
Length Function	2-51
LET Directive	2-52
LIST Directive	3-16
List Function	2-54
LOAD Directive	3-18
LOCK Directive	5-17
Logical AND Function	2-56
Longitudinal Redundancy Check	2-57
LRC	2-57
LST	2-54
 Manual Organization	 1-1
Masking	6-13
Mass Memory	5-1
MERGE Directive	3-19
Mnemonic Constants	6-16
Mnemonic Controls	6-9
Mnemonic Device Control	6-10
MOD	2-58
Modulo Function	2-58
 NOT Function	 2-59
NUM	2-60
Numeric Constants	2-3
Numeric Data	2-2
Numeric Dimension Directive	2-31
Numeric Evaluation	2-6
Numeric Expressions	2-6

Numeric Functions	2-7, 2-60
Numeric Subscripts	2-3
Numeric Variables	2-3
Numeric Verification	6-6
ON GOSUB Directive	4-14
ON GOTO Directive	4-16
OPEN Directive	5-18
Output Directives	6-1
Output Formatting	6-13
Pack Function	2-61
PCK	2-61
PGM	2-68
PGN	2-66
POS	2-62
Position Function	2-62
Position Handling	6-9
Position Specifiers	6-8
PRECISION Directive	2-64
PRINT/PRINT RECORD Directive	6-34
PROGRAM	5-20
Program Control	4-1
Program Control Directives	4-4
Program File Directive	5-20
Program Files	5-2
Program Mode Processing	3-2
Program Name Variable	2-66
Program Processing	3-2, 4-1
Program Size Variable	2-67
Program Statement Function	2-68
Program Statements as Data	2-12
Programmable Numeric/String Functions	2-28, 2-70
Programs	3-2
PSZ	2-67
PUB	2-72
Public Programs	7-1
Public Programs Directives	7-2
Public Programs Directory Function	2-72
Public Programs, Restrictions	7-2
PUT Directive	6-39
QUO	6-16

Random Access Memory	5-1
READ/READ RECORD Directive	6-41
Record Specification	6-2
RELEASE Directive	3-22
REM	3-23
Remarks Directive	3-23
REMOVE Directive	6-45
RENAME Directive	5-22
RESERVE Directive	5-23
RESET Directive	3-24
RETRY Directive	4-17
Return Address Stack	4-2
RETURN Directive	4-18
RUN Directive	3-25
SAVE Directive	5-24
Sector Size Function	2-73
SEP	6-16
Serial Number Variable	2-78
SETDAY Directive	2-74
SETERR	4-19
Set Error Directive	4-19
SETESC	4-20
Set Escape Directive	4-20
SETTIME Directive	2-75
SETTRACE Directive	3-27
SGN	2-76
Sign Function	2-76
Significant Digits	2-3
SIZ =	6-4
SMC BASIC Utilities	9-1
SORT	5-26
Sort File Directive	5-26
Sort Files	5-3
Special Character Representation	6-15
Special Code Representation	6-15
Special Processing Directives	2-11
SSN	2-78
SSZ	2-73
START Directive	3-28
Statement Conversion	2-12
STOP Directive	3-30
STR	2-77

String Data	2-4
String Dimension Directive	2-33
String Expressions	2-7
String Functions	2-7, 2-9, 2-77
String Subscripts	2-5
String Verification	6-7
Syntax Conventions	1-2
Syntax Errors	8-1
System and Task Variables	2-9
SYS	2-79
System Directives	3-1
System Serial Number Variable	2-78
System Variable	2-79
TABLE	6-47
Task Control Block Function	2-80
Task Memory Parameters Function	2-81
Task Status Function	2-82
Tasks	3-1
TCB	2-80
Termination Status Message Variable	2-83
Thoroughbred/OS Capabilities	1-1
Thoroughbred/OS Utilities	9-1
TIM	2-84
TIM =	6-4
Time Variable	2-84
Translation Table Statement	6-47
TSK	2-81
TSK(0)	2-82
TSM	2-83
UNLOCK Directive	5-28
Unpack Function	2-85
Untrapped Errors	8-2
UPK	2-85
VFU Loading	B-1
WAIT Directive	4-21
WRITE/WRITE RECORD Directive	6-50
XOR	2-37



T66-011567-RM01